Certification Guide

Sair Linux and GNU Certification®

Level II: Apache & Web Servers

- Includes 100 sample exam questions for assessing skills and improving test results
- Covers all system administration issues for a Linux and GNU enterprise system

Sair Development Team

SAIR LINUX GNU CERTIFICATION



Sair Linux and GNU Certification®
Level II: Apache and Web Servers

Sair Development Team

Wiley Computer Publishing



John Wiley & Sons, Inc.

NEW YORK • CHICHESTER • WEINHEIM • BRISBANE • SINGAPORE • TORONTO



Sair Linux and GNU Certification®
Level II: Apache and Web Servers

Sair Development Team

Wiley Computer Publishing



John Wiley & Sons, Inc.

NEW YORK • CHICHESTER • WEINHEIM • BRISBANE • SINGAPORE • TORONTO

Publisher: Robert Ipsen
Editor: Cary Sullivan
Assistant Editor: Christina Berry
Managing Editor: Marnie Wielage
Text Design & Composition: Benchmark Productions, Inc.

Designations used by companies to distinguish their products are often claimed as trademarks. In all instances where John Wiley & Sons, Inc., is aware of a claim, the product names appear in initial capital or all capital letters. Readers, however, should contact the appropriate companies for more complete information regarding trademarks and registration.

Copyright © 2001 by Sair Development Team. All rights reserved.

Published by John Wiley & Sons, Inc.

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 750-4744. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 605 Third Avenue, New York, NY 10158-0012, (212) 850-6011, fax (212) 850-6008, E-Mail: PERMREQ@WILEY.COM.

This publication is designed to provide accurate and authoritative information in regard to the subject matter covered. It is sold with the understanding that the publisher is not engaged in professional services. If professional advice or other expert assistance is required, the services of a competent professional person should be sought.

This title is also available in print as ISBN 0-471-40537-X. Some content that appears in the print version of this book may not be available in this electronic edition.

Contents

Acknowledgments xi

Introduction xiii

Part One Knowledge Matrix 1

Chapter 1 Installation and Configuration 3

Objectives 3

Theory of Operations 3

History of Apache 4

Apache Today 5

How Does Apache Work? 5

How to Obtain Apache 10

Overview of Content Negotiation 11

Base Systems 18

Preparing Linux 18

Introduction to Packages 21

Installation 29

System Utilities 34

The httpd Daemon 34

Setting Up Apache 45

Chapter 2 System Administration 55

Objectives 55

Theory of Operation 56
Being a Webmaster 56
Preparing Apache 58
Introduction to Virtual Hosting 63
Introduction to Apache Modules 64
Introduction to the Apache API 76
Introduction to Logging 83
Base Systems 84
Multiple Daemons 84
Configuration 85
Number of httpd Processes 88
Alias 89
CGI Scripts 89
How to Configure CGI 90
Apache Initialization 91
Log Files 93
Log File Formats 95
Shells and Commands 98
Benchmarking 98
System Utilities 100
Creating CGI Scripts 100
Performance Monitoring 107
Some Good Log Analysis Tools 109

Chapter 3 Networking 111
Objectives 111
Theory of Operation 111
What Is TCP/IP and How Does Apache Use It? 112
What Is HTTP? 113
Multiple Hosts 115
Base Systems 116
Virtual Hosting 117
Directing the Request to a Virtual Host 118
Single Daemon/Virtual Hosting 118
IP-Based Virtual Hosting 119
Name-Based Virtual Hosting 122
Shells and Commands 125
URL Rewriting mod_rewrite 125

Chapter 4 Security 133
Objectives 133
Theory of Operation 133
Security Concerns 134

Security Policies 134
Authentication 136
Securing Apache 137
Vulnerabilities 138
Hostile Programs 139
Security Issues with CGI 142
The Apache Proxy Server 143
Firewalls 146
Password Protection 153
Base Systems 154
Apache, Users, and Groups 154
Permissions 155
Access Control 156
Setting Up the Apache Proxy 159
Security Fundamentals 164
User Access Control 167
Enabling Content from Home Directories 170
Access Directives 172
Defining within httpd.conf 173
Shells and Commands 174
Checksums 174
Password Authentication 176
System Utilities 177
Server-Side Includes 177
XSSI 179
ModSSL versus Apache+SSL 183

Chapter 5 Troubleshooting 187
Objectives 187
Online Troubleshooting Resources 187
Tracking Down an Apache Core Dump 188
Some Useful Sites 189
Configuration Issues 190
Logging Problems 190

Part Two Labs and Exercises 193

Lab I Installation 195
Purpose 195
Theory 195
Lab Exercises 196

	Downloading Modules	196
	Preinstallation Query	197
	Package Installation	198
	Basic Server Setup	198
	Questions	199
	Answers	200
	Advanced Questions	200
Lab II	**Install Apache+SSL**	**201**
	Purpose	201
	Theory	201
	Lab Exercises	202
	Downloading the Apache server	202
	Compile Apache with mod_ssl Support	203
	Verify That Apache Was Compiled with mod_ssl	204
	Test the Sample Page in a Web Browser	205
	Questions	205
	Answers	206
	Advanced Questions	206
Lab III	**Configuring Apache to Perform Common Tasks**	**207**
	Purpose	207
	Theory	207
	Lab Exercises	210
	Questions	210
	Answers	211
	Advanced Questions	211
Lab IV	**Create a Simple CGI Script**	**213**
	Purpose	213
	Theory	213
	Lab Exercises	214
	Create a Basic CGI Script	214
	Questions	215
	Answers	216
	Advanced Questions	216
Lab V	**Configure and Run mod_auth_mysql**	**217**
	Purpose	217
	Theory	217
	Setting Up the MySQL Database	218
	Setting Up Apache	218
	Lab Exercises	218

	Student Resources	221
	Questions	221
	Answers	221
	Advanced Questions	221
Lab VI	**Apache and Tomcat**	**223**
	Purpose	223
	Theory	223
	Lab Exercises	224
	Questions	225
	Answers	225
	Advanced Questions	225
Lab VII	**Configuration of a Proxy**	**227**
	Purpose	227
	Theory	227
	Installing mod_proxy	228
	Configuring httpd	228
	Configuring the Client	229
	Lab Exercises	229
	Installing mod_proxy	229
	Configuring httpd	229
	Configuring the Client	230
	Questions	230
	Answers	231
	Advanced Questions	231
Lab VIII	**URL Rewriting**	**233**
	Purpose	233
	Theory	233
	Lab Exercises	236
	Questions	237
	Answers	237
	Advanced Questions	237
Lab IX	**Create a Custom Log for Apache**	**239**
	Purpose	239
	Theory	239
	Lab Exercises	241
	Questions	242
	Answers	242
	Advanced Questions	242

Lab X **Benchmark Your Server** **243**
Purpose 243
Theory 243
Lab Exercises 245
Questions 247
Answers 247
Advanced Questions 247

Part Three Practice Questions and Answers **249**

Practice Questions **251**

Answers **273**

Glossary **281**

Index **290**

Acknowledgments

With the growing body of certificate holders and with the development of Level II, the team effort required to implement the Sair Linux and GNU curricula has been amazing. As a result, our acknowledgment list continues to grow to reflect the greater involvement in the open source community:

Bill Patton, Steering Committee Chair, Compaq; Cheryl Foiles, Productivity Point International; Mark Muth, Prometric; Dan Kusnetzky, International Data Corporation; Kevin Whittier, MSC Software; Timothy Ney, Free Software Foundation; Eric S. Raymond, Consultant and Open Source Software Author; Richard Stallman, Founder of the GNU GPL and Free Software Movement; Tim Angle, University of Mississippi; Cary Sullivan, John Wiley and Sons Inc.; Jon "Maddog" Hall, Linux International; Bill Noyes, Magellan Group, Inc.; Evan Blomquist, Technical Trainer; Doug Dickerson, Compaq; Stephen Solomon, Course Technology; Bruce Perens, Hewlett-Packard; Stuart Trusty, Linux Labs; Bill Arvidson, Mission Critical Linux; Dan Greening, Motorola Computer Group; Bryan Ochs, New Horizons Computer Learning Centers; Mark Langston, The SysAdmin Company; Deb Murray, Uniforum; Kit Cosper, VA Linux Systems; Ken Kousky, Wave Technologies; Dr. James Stanger, ProSoft Training.com; Steven Wright, Caldera Systems; Paul Wildrick, CyberstateU.com; Tina Bush, Element K; Don Corbet, Team Linux Corporation; Harald Bertram, AG. Abt. Training; SuSE Linux Solutions; John Terpstra, Caldera; Jeremy Siadal, Intel Corporation; Kerry Hodgins, Corel Corporation; Karen Letain, Wave Canada; and Jim Lacey, Linuxcare.

Continuing the process of shaping mountains of trivia into some semblance of organization and developing that information so that it can be communicated quickly and effectively is the task of the Sair Development Team. These are the people who assisted with the production of the Apache material:

President and CEO: Dr. Tobin Maginnis
Managers: Ross Brunson, Trish W. Kemerly, Carlos Pruitt, and Paul Tate
Assistant manager: Albert Phillips
Contractors: Ancilla Allsman, Joseph Cheek, Mike Hartley, Tony Inson, Alex Larson, Rasmus Lerdorf, Shawn McKenzie, Martin Pool, Ed Riddle, Andrew Scott, Troy Vitullo, and Susan Wood
Technical contributors: Stephen Agar, David Austin, Tim Bauer, Jerald Jones, and Omar Wilson
Technical editor: Wesley Duffee-Braun
Editors: Annie Current, Summer Hill, Jamie Murphey, and Samantha Rayburn
Quality assurance: Jenna Johnson and Nikki Andersen
Graphics: Wesley Duffee-Braun and Leah Riley

The Sair Development Team would also like to thank the rest of the employees at Sair Linux and GNU for their support—albeit technical, promotional, or otherwise—and for contributing to the Linux community through their efforts:

Managers: Bob Buntyn, Les Driggers, Hollis Green, Leigh Jennings, Alex Lundy, and Lenny Sawyer
Assistant managers: Jimmy Palmer and Elizabeth Bonney
Technical contributors: Eric Blankenship, Beau Bourgeous, Jeff Britt, Brett Brown, Jayaprakash Chilumula, Joann Chong, John Furr, Stephen Goertzen, Scott Hicks, Jacob Jenkins, David Kearns, Rebecca Love, Chris Mavromihalis, Michael McGuire, Damir Mehmedic, Chance Mobley, Ken Montgomery, Rufus Peoples, Ben Pharr, Evan Rouse, Tyler A. Simon, Robert Thompson, Sunder Upadhyay, William Vaughan, Sudharshan Vazhkudai, James Walker, James Webster, and Dana Wilson
Test development: Tammy Betts, Michael McGuire, and Richard Swinney
Programmers: Michael Broadwater, Michael Calvi, Chris Mavromihalis, Rufus Peoples, William Vaughn, and Jeremy Webster
Web and database: Shafi Al-meher, Steve DeVries, Nileshwar Dosooye, Janakiram Govindaraju, Mark Ketcham, Arlene Pereira, Kubenthiran Ramanathan, Errol Sayre, and Julie Seay
Systems: Patrick Hood, Ross Reed, and William Taylor
Production: Brandi Bailey and Scott Rains
Sales and marketing: Wendy Chambers, Scott Thompson, Shellie Ross, Karan Mullen, Haley Teague, Beth Odum, Shelley Robbins, Elise Knapp, Ashly Ray, and Lori Redding

Again, thanks to John Wiley & Sons Computer Publishing, especially Cary Sullivan, for their foresight, effort, and belief in the future of free, open source software.

Introduction

Welcome to the Level II study guide series for the Sair Linux and GNU Certified Engineer. This is the Apache study guide for Sair Linux and GNU Certification, Exam 3X0-202. The Apache study guide is one of the optional or elective subjects for the Sair Linux and GNU Certified Engineer (LCE) certificate.

Requirements for the Sair Linux and GNU Certified Engineer (LCE) include passing the Core Concepts and Practices exam, 3X0-201, plus three additional exams from other elective subjects. In addition to this study guide, other possible elective areas include Sendmail and mail system components, Samba and resource-sharing components, and PHP and scripting. See www.linuxcertification.com for a complete listing of available courses. Each course consists of a minimum of 32 classroom contact hours, making the LCE equivalent to an additional 128 hours of classroom contact beyond the LCA. We see achievement of the LCE to be evidence of the ability to perform as an advanced Linux systems specialist, as a system administrator supervisor, or in some other form of Linux system management.

Exams are offered at Prometric and VUE testing centers. Prometric offers the Sair exams at any one of its 3,500 testing centers in 141 countries. To take a test, simply call the Prometric registration line at 1-888-895-6717. Ask the customer service representatives about the 3X0 series of exams, and they will answer any test-taking questions related to the test, describe available local testing centers, and, if requested, schedule an exam. It is also possible to register for tests online at www.prometric.com. VUE also offers the Sair exams at any one of its 2,500 testing centers in 110 countries. To take a test, call the VUE registration line at 1-952-995-8800. Ask the customer service representatives about the 3X0 series of exams, and they will answer any test-taking questions related to the test, describe available local testing centers, and, if requested, schedule an exam. It is also possible to register for tests online at www.vue.com/sairlinux.

Preparing for test content is paramount in criteria-based certification tests, and Sair takes pride in the coordination of test topics from the Knowledge Matrix, objectives, competencies, study guides, and exams. These topics specify exact criteria that the candidate must meet. The tests directly measure these topics, and results are reported with a detailed summary for each of these six areas: theory of operation, base system, shells and commands, system utilities, applications, and troubleshooting.

Each exam consists of 50 questions that can be answered, reviewed, and changed for up to 60 minutes. Typical test takers use about 45 minutes to complete the exam. Successful completion of each test requires 74 percent correct, or 37 correct answers. Unlike tests that assign a rank to a test taker relative to all other test takers, Sair emphasizes mastery of material. Results of the test directly inform the candidate of the mastery level in each area, allowing each student to focus future studies on areas of relative weakness. Prospective employers can also use the detailed summary to evaluate job applicants' or employees' areas of strength. Sair is unique in this regard. Some other certification examinations supply the test taker with only a relative score of some type, without a raw score, percentage correct, or other measure for evaluating his or her performance. The effect of this practice is to leave the test taker who fails without guidance as to how to prepare for reexamination.

Minimum Candidate Requirements

The Sair Linux and GNU tests were not designed for the novice computer user. It is assumed that the candidate has approximately two years of computer experience and has experience in the configuration of one or more operating systems. For example, the candidate should be familiar with basic hardware concepts, such as CPU, cache, memory, interface adapters, hard disks, and networks. The candidate should also be familiar with basic operating system concepts, such as booting, file access, and device drivers. Finally, the candidate should know basic commands and the use of a Unix-type editor, such as Joe, Pico, Vi, or Emacs.

Knowledge Matrix

The test is based on the Apache Knowledge Matrix shown at www.linux-certification.com. Examination topics are listed here. Note that while the major topics listed in the Knowledge Matrix will not change, some subtopics may be added as needed to update the material. Please check the Sair Web site for any additions to this list.

Apache and Installation and Configuration

Theory of Operation

1.1.10 History of Apache

1.1.20 Apache Today

1.1.30 How Does Apache Work?
- A. Web server concepts
- B. Apache conventions
- C. Directives
- D. Handlers
- E. Logging
- F. Modules
- G. Lynx

1.1.40 How to Obtain Apache

1.1.50 Overview of Content Negotiation
- A. Type
- B. Encoding
- C. Language
- D. Multiviews
- E. Browsers and HTTP
- F. Style sheets
- G. Java
- H. JavaScript

Base Systems

1.2.10 Preparing Linux
- A. RAM
- B. Hardware
- C. Hard drive
- D. Kernel
- E. File handles and inodes

1.2.20 Introduction to Packages
- A. RPM security tasks
- B. Locate security updates
- C. Add modules via RPMs
- D. Start Apache
- E. Verify functionality locally
- F. Verify functionality over a network
- G. Source RPMs

1.2.30 Installation
- A. Compiling
- B. Dynamic shared objects
- C. Installing
- D. Testing before installing

System Utilities

1.4.10 The httpd Daemon
- A. Defaults
- B. Customization
- C. Modules and SSL
- D. Module performance and functionality
- E. Enabling modules
- F. Configuration files
- G. Configuring Apache logs
- H. Alias
- I. Modules
- J. Configuration methods
- K. Starting httpd at boot

1.4.20 Setting Up Apache
- A. Starting and stopping Apache
- B. Default index
- C. Basic configuration
- D. Port 80
- E. ServerType stand-alone
- F. StartServers 5
- G. MinSpareServers 5 and MaxSpareServers 10
- H. MaxClients 150
- I. MaxRequestPerChild 0
- J. Basic configuration
- K. Document directory configuration
- L. Options directive

Apache and System Administration

Theory of Operation

2.1.10 Being a Webmaster
- A. Tuning

2.1.20 Network Access Installation
- A. Allow and Deny

B. Max Clients
C. TCP/IP version
D. Hostname lookups
E. FollowSymLinks and SymLinksIfOwnerMatch
F. AllowOverride
G. Content negotiation
H. Process creation
I. Process death
J. KeepAlive

2.1.30 Introduction to Virtual Hosting
A. Directories
B. Network addresses

2.1.40 Introduction to Apache Modules
A. What is CGI?
B. When to use CGI
C. PHP – mod_php4
D. MySQL
E. mod_perl

2.1.50 Introduction to the Apache API
A. Modules
B. Introduction to phases
C. Phases in detail
D. The future: Apache 2.0

2.1.60 Introduction to Logging

Base Systems

2.2.10 Multiple Daemons

2.2.20 Configuration
A. Running multiple daemons
B. Multiple daemon verification

2.2.30 Number of httpd Processes

2.2.40 Alias

2.2.50 CGI Scripts
A. ScriptAlias directive
B. AddHandler directive

2.2.60 How to Configure CGI
A. Enable within certain directories

2.2.70 Apache Initialization

2.2.80 Log Files
- A. /var/apache/logs/access_log
- B. host
- C. ident
- D. authuser
- E. date
- F. request
- G. status
- H. bytes

2.2.90 Log File Formats
- A. Design your own log files format

Shells and Commands

2.3.10 Benchmarking
- A. ab

System Utilities

2.4.10 Creating a CGI script
- A. Content of CGI output
- B. Server environment
- C. Form handler
- D. Basic redirect

2.4.20 Performance Monitoring
- A. mod_status
- B. ExtendedStatus
- C. server-info

2.4.30 Some Good Log Analysis Tools
- A. Webalizer
- B. Analog
- C. Wusage
- D. Summary
- E. Logresolve

Apache and Networking

Theory of Operation]

3.1.10 What Is TCP/IP and How Does Apache Use It?
- A. HTTP
- B. Headers

3.1.20 What Is HTTP?
A. HTTP/1.1

3.1.30 Multiple Hosts

Base Systems

3.2.10 Virtual Hosting
A. Configuring separate daemons

3.2.20 Directing the Request to a Virtual Host

3.2.30 Single Daemon/Virtual Hosting

3.2.40 IP-Based Virtual Hosting
A. device
B. aliasnumber
C. address

3.2.50 Name-Based Virtual Hosting
A. Virtual hosting with one IP address
B. Virtual hosting using mixed methods

Shells and Commands

3.3.10 URL Rewriting mod_rewrite
A. CondPattern TestString

Apache and Security

Theory of Operation

4.1.10 Security Concerns

4.1.20 Security Policies
A. hosts.allow and hosts.deny
B. motd and issue files
C. U.S. encryption export laws

4.1.30 Authentication

4.1.40 Securing Apache
A. Apache user

4.1.50 Vulnerabilities
A. Trojan horses
B. Viruses
C. Worms
D. Spoofing
E. Buffer overruns

4.1.60 Security Issues with CGI
- A. System calls
- B. Buffer overruns

4.1.70 The Apache Proxy Server
- A. Advantages of the Apache proxy
- B. Obtaining the Apache proxy
- C. Obtaining documentation
- D. mod_proxy

4.1.80 Firewalls
- A. Types of firewalls
- B. Proxy server firewalls
- C. IP masquerading proxy servers
- D. Firewalls and network architecture
- E. Securing the firewall machine
- F. What to do if attacked

4.1.90 Password Protection

Base Systems

4.2.10 Apache, Users, and Groups

4.2.20 Permissions

4.2.30 Access Control
- A. AllowOverride
- B. Order, Allow, and Deny
- C. Testing
- D. Anonymous access

4.2.40 Setting Up the Apache Proxy
- A. Proxy-specific directives
- B. Server-side configuration

4.2.50 Security Fundamentals
- A. Permissions
- B. Scripting
- C. SuExec
- D. Matrix of ideal permissions

4.2.60 User Access Control
- A. Common access controls

4.2.70 Enabling Content from Home Directories
- A. UserDir html

4.2.80 Access Directives
- A. AuthType
- B. AuthName
- C. AuthName "Private Documents"

4.2.90 Defining within httpd.conf
- A. Defining within control files

Shells and Commands

4.3.10 Checksums
- A. PGP and checksums

4.3.20 Password Authentication
- A. User
- B. Group

System Utilities

4.4.10 Server-side Includes
- A. Basic commands
- B. Variables

4.4.20 XSSI
- A. Shell commands
- B. Including files
- C. Executing scripts
- D. Embedding XSSIs
- E. Conditional statements

4.4.30 ModSSL vs. Apache+SSL
- A. SSL—mod_ssl
- B. SSL—Secure Sockets Layer (SSL)
- C. Implementing SLL in Apache
- D. Legal issues

Apache and Troubleshooting

5.1.10 Online Troubleshooting Resources

5.1.20 Tracking Down an Apache Core Dump

5.1.30 Some Useful Sites

5.1.40 Configuration Issues

5.1.50 Logging Problems



Knowledge Matrix

CHAPTER
1

Installation and Configuration

Objectives

- Define the evolution of Apache.
- List the Apache levels of configuration.
- Define Apache configuration options and methods.

Theory of Operations

Almost two-thirds of the Web servers on the Internet use Apache (see Figure 1.1). Apache is common for the same reason that screwdrivers and crescent wrenches are common—doing a job right takes the proper tool. Apache is an excellent tool for World Wide Web hosting.

Although Apache may not perform well in some benchmark tests, it performs extremely well in the field. It is a fully capable, industrial-strength Web server that is able to compete with other servers that are much more costly to purchase or license. It is extremely stable and has been heavily tested over the years in many different environments and platforms. The

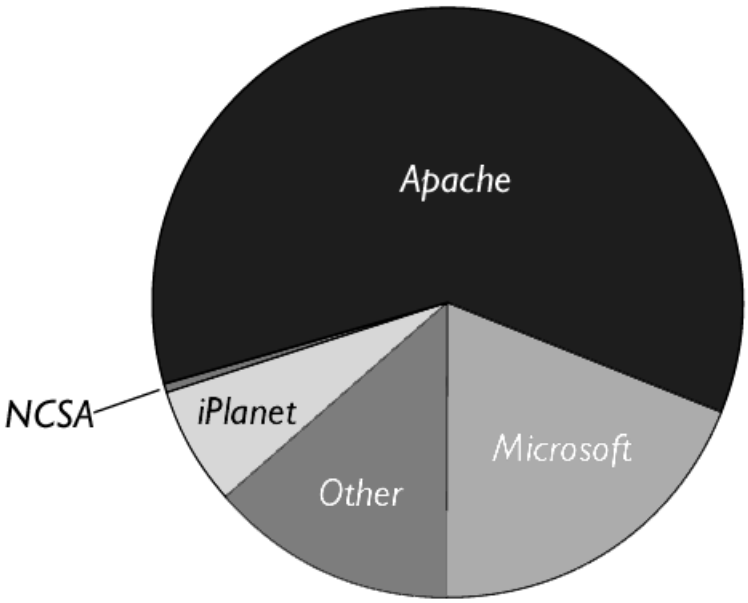


Figure 1.1 Survey of Web server prevalence on the Internet.
www.netcraft.com/survey

developers of Apache emphasize performance above all else. Webmasters will appreciate the richness of features that can be implemented on a site served by Apache.

History of Apache

Apache is based on NCSA's `httpd`, a daemon that was first used in the early years of the Internet. Rob McCool was the creator of the NCSA Web server and continued his work on it until 1994. No single programmer took his place, but the server continued to grow in popularity. After his departure from the project, incompatibilities between versions began to develop. Web administrators had to make changes on their own if they wanted to fix bugs or add features. Eventually, a group of administrators began working together to regain control of the project. Their efforts created a single path through which patches and enhancements to `httpd` could be submitted. The name of the project is derived from its origins: a series of patches applied to the original `httpd` daemon. The project came to be known as "a patchy server," and from there, Apache server.

Apache Today

Apache possesses a level of complexity that easily surpasses some operating systems. Although it is not necessary to learn every feature to gain a functional understanding of Apache, an introduction to the overall design of the environment and methods used is a good start.

The purpose of this section is to describe some of the configuration styles that Apache uses and to provide an overview of some commonly encountered server features. Later in the course, when topics are described in more detail, a context is developed on which to base further learning. Ultimately, the goal of this book is to teach the user to set up an Apache server.

The user can also go to the Apache Web site, www.apache.org/docs/, and study the full documentation in depth. The online user documentation for Apache is one of the most well-written and accessible sources of information for any project.

How Does Apache Work?

Apache is a monolithic Web server. It looks to centralized configuration files for all parameters that are not compiled. Apache works much like the Linux kernel. It forks off child processes, accepts loadable modules, and operates via user-specified configurations.

Web Server Concepts

On a system functioning as a Web server, the server process listens for incoming requests on a specified port of the network interface and responds to them appropriately. Originally, a Web site consisted of a directory tree of static documents, which would be served on request. Special functions could be handled by CGI scripts that are executed on request.

Over the years, people have developed variations of this basic function, requiring a rethinking and retooling of the original Web server definition to the point that today's Web users and authors expect a certain amount of active participation by the server. Dynamic content is everywhere now; today's server must not only provide a Web page, it must create it on user demand. Although the basic process is the same, expected enhancements place much more demand on a server than what was required when static content was the only possibility.

Apache Child Processes

Apache works by splitting itself and having the Apache copy handle the process. Child processes are the processes that actually dole out content. When Apache receives a request, it chooses one of its idle child processes to handle the request. When the child is done, it returns to an idle state, ready for more work.

Every child completes its list of requests before dying. Killing a child process after it completes its list of requests also kills any side effects of those processed requests, such as memory leaks. The user can determine the number of children Apache spawns at one time, as well as their life span. By default, each child handles an unlimited number of requests.

Apache Conventions

A good way to learn how Apache functions is through its configuration. The first level of configuration occurs when Apache is compiled. RPM- and Debian-based installations decide many things by default, but when compiling from source these options can be set:

- Whether Apache uses Dynamic Stored Objects or statically linked modules (modules will be described in further detail later in the chapter)
- If static modules are chosen, tell Apache which modules to compile into the server at this point
- Default location
- Layout of Apache's files and directories

The next level of configuration occurs in `httpd.conf` after the installation. The `httpd.conf` is a global preferences file that controls all aspects of the server, including the following:

- Who can access the Web server
- What dynamic modules the server loads
- Where certain functions occur
- What kind of information Apache will log
- The location of the virtual Web sites Apache controls
- What content is allowed

The last level of configuration may occur in `.htaccess` files, depending on what is stated in `httpd.conf`. These are small, local files that contain specific orders that apply to the directories in which they reside.

The `httpd.conf` file contains more than three-fourths of the Apache configuration options. It allows administrators to configure Apache to an optimal level at performance. This is possible because, based on users' needs, Apache offers many levels of control through the `httpd.conf` file. These levels are called scopes. A scope can apply to an entire system, a virtual host, an absolute directory path, or even a single file. Wild cards can be used with scopes to give greater freedom. Containers are used to define each scope.

The following are the four main containers:

```
<Directory /path>
# This container applies to the filesystem location of a directory.
# For example, /home/user/public_html would point to user's public_html directory.
Directive
</Directory>

<Location /~*>
# Location applies to a url. For example ~/user would point to user's public_html
directory.
Directives
</Location>

<VirtualHost host_name>
# VirtualHost applies to a virtual web site you are hosting, such as mysite.com
Directives
</VirtualHost>

<Files file_name>
# Applies to a file name, no matter where it is located. If file_name were
index.html, then all instances of index.html (no matter what their location) would be
affected by your directives.
Directives
</Files>
```

Directives

A directive is a command given to Apache that controls its behavior. Directives are a single line of text with the following format:

```
Directive [option [option]]
```

As shown, there is more than one option for your key word. Although there can be more than one directive in a given scope, each directive must occur on its own line. Directives that do not reside in any container apply to all scopes.

Handlers

When a user requests a file, Apache decides what to do with it. For example, if the user accesses a file called `hiccup.cgi`, Apache must know to execute that file as a CGI script and not try to display it as an HTML file. Handlers help you direct Apache to the correct helper application. For example, add a handler called `cgi-script` and apply it to all files with an extension of `.cgi` at the end.

Logging

Apache logs almost everything that happens to it, from errors to http transactions and even cookies. The user can set the level of logging and what is logged in the `httpd.conf` file.

Modules

Modules are chunks of software (usually written in Perl or C) that allow customization of Apache. For instance, for Apache to use Server Side Includes, the user must install that module.

Even though Apache was originally developed with this concept in mind, it was not available when the first versions of Apache were being developed. Still, because it was already being planned at that time, integration of dynamic module loading into the Apache server came relatively easily.

In early versions of Apache, modules were statically linked into the `httpd` application during compilation. The specific configuration of a server was defined during the configure process before compilation. Using this method, many helpful modules were developed over the years, making a tremendous amount of customization available for the Apache server. Since the release of Apache version 1.3, it is possible to compile the server and the modules to support dynamic loading. With this option compiled, the configuration of the server can be changed by simply restarting the server after editing the configuration file (rather than recompiling the server). Different instances of the server, with different features, can be started at separate times or simultaneously by using different configuration files running the same base installation.

Today, only two modules absolutely must be compiled into the server: `http_core.c`, which provides the core functionality of the server, and `mod_so.c`, the module that supports Dynamic Shared Objects (DSO).

Static Module Advantages

On occasion, it may be desirable to compile other modules into the server so that they are always available, giving a slightly faster response and guaranteeing that a specific version of the module is present at all times.

Static Module Disadvantages

One disadvantage to compiling a module statically into the server is that it continues to take up space within the server, even if its features are not used. Another disadvantage is that upgrading the static module requires the user to recompile the server.

DSO Advantages

For the most part, the benefit of being able to add or remove a feature or update a feature to a newer version without having to recompile the server is tremendously helpful.

DSO Disadvantages

When using DSO files to configure Apache, the server is approximately 20 percent slower at startup because the module loader must find all of the modules, load them, and resolve relocation symbols. This is not a serious problem, though, because startup is an infrequent event.

On some platforms, the server may run about 5 percent slower because relative addressing of position-independent code is more complicated and slower than absolute addressing.

There are a few other potential problems with implementing DSO modules, but these mainly apply to platforms other than Linux. See the Apache Web site for further details.

DSO Files

Since the advent of DSO files, it is much more common for distributions to include the Apache server with binaries already installed and to make installation of the source code optional. This means that the full set of files necessary to compile a new module acquired from another source is not initially present on the system. So that this would not be a problem, the authors developed the APache eXtenSion (APXS) product. This is a Perl program created at installation that gives access to all Apache header files,

as well as platform-dependent compiler and linker flags. The user is then able to compile Apache modules without the Apache source tree and without struggling with platform-dependent linker and compiler flags.

APXS Usage

The following code gives a generic example of APXS usage, which builds a module that can be dynamically loaded into the Apache server:

```
$ cd /path/to/the_module
$ apxs -c mod_new_module.c
$ apxs -i -a -n foo mod_new_module.so
```

Browsers

There will be labs during this course in which you need to simulate what a client sees. To aid in this simulation, you will need an X Window system install, as well as a few extra Web browsers, namely Netscape 4.7x, Mozilla, and a recent beta of Opera. Installation of these browsers, in addition to the X Window system, is recommended for this course.

It is highly unlikely that a Web server would ever need a graphical interface or graphically enhanced browsers because they tend to make use of valuable resources. They will be used throughout this course, however, for testing, examples, and labs. In addition, graphical interfaces will be recommended, if not required.

NOTE **Throughout the course we use Lynx as a test tool. We recommend this program over Netscape due to the resource overhead that Netscape adds to your machine. If you prefer a GUI tool, use Netscape instead of Lynx.**

To download Lynx go to www.slcc.edu/lynx/realease. For more Lynx information go to:

www.lynx-browser.org

www.trill-home.com/lynx.html

www.slcc.edu/lynx/release2-8-2/lynx2-8-2/lynx_help/lynx_help_main.html

How to Obtain Apache

First, obtain the latest source code from http://httpd.apache.org/dist/.

At the time of publication, the latest stable release was in the tarball `apache_1.3.19.tar.gz`. Of course, the `rpm` or `deb` file provided by the distribution can be used as well, but those installation methods are not covered here. We recommend removing any previously installed packages prior to installing from source. For example, use `rpm -e` on the packages identified with `rpm -qa |grep apache` (or `apt-get --purge remove apache` for Debian-based installs). Be sure to remove the dependencies as well. If not installing from source, use the packages relevant to the distribution and perform an upgrade. The remainder of this chapter will be based on the source installation described. Therefore, paths and file locations must be translated for each particular installation.

Installing from source is an option for any distribution, and this method will be discussed in this text. Once all previous installations are removed, unpack the tarball. Do not worry about the remaining configuration files; however, they may need to be moved because the source installation may not overwrite some of them. We use `/usr/src` as our point of origin, but any directory may be used for the following steps.

Overview of Content Negotiation

Modern browsers allow their users to indicate many preferences for files that they will receive from a Web server. This includes content type (for example, a preference of `tar.gz` over `.zip` files for compressed files), preferred image formats, and preferred language.

To handle these preferences, each document has its own set of variants. The way in which these variants differ is called the document's dimensions. Apache can negotiate along the dimensions of type, encoding, and language.

Type

Every document has a type. An example type is `text/html`. If a Web browser tells Apache that it can accept only a given type of document, Apache will serve only that type to the browser.

Encoding

If a resource is compressed or otherwise not in its final, usable format, it is encoded. The Web transport standard recognizes only the compress and gzip encodings currently. There are other, experimental encodings that the

standard can handle now. Their encoding is preceded with `x-`, as in `x-stuffit` for Stuffit archives.

```
AddEncoding x-stuffit
```

Language

For once, when referring to language in computer terms, we really mean human spoken and written languages. The language of a file is designated by adding a file extension to it, then using Apache's `AddLanguage` directive, `AddLanguage MIME extension [extension].`

An example of a file or document name using language extensions is:

```
index.html.fi
dummy.en.zip
```

The `httpd.conf` file that defines the file types specified by the file syntax would look like the following:

```
Addlanguage    en.en
Addlanguage    fi.fi
```

These two directives specify that any file with the `en` or `fi` language tag located at the end of a file name will be classified as English or Finnish, respectively. They also show that the tag may be one dot before the last section of the file name.

Choosing Document Parameters

A Web server can handle multiple variable dimensions, as can any given client. The client preferences are listed in order of quality. Values for quality range from 0.000 (never show the given dimension) to 1.000 (the most preferable dimension).

For example, a browser might send the following in its header:

```
Accept-Language: en; q=1.0, no; q=0.7
```

This means the browser accepts English first and Norwegian second.

The server can also have quality values on multiple variants. When it has to make a decision between multiple variants that are equally acceptable to the client, the server uses its own quality preferences.

MultiViews

`MultiViews` instruct Apache to look for a document that is the best match for a client's preferences. For example, if a client requests http://www.linux.org/index and no file exactly matches that request, Apache scans the directory involved for variants of `index`. Apache finds all files with similar names, then checks the client's content preferences (Type, Encoding, and Language) to find a suitable match.

NOTE **Scanning greatly increases disk I/O activity, slowing down the Web server.**

Because of the high I/O bandwidth inherent with `MultiViews`, when specifying `Options All`, `MultiViews` are off. To turn `MultiViews` on, one must specify:

```
Options +MultiViews
```

for a given scope. This adds `MultiViews` to the existing options.

For more information about content negotiation, visit the following two sites:

http://httpd.apache.org/docs/content-negotiation.html

http://httpd.apache.org/docs/mod/mod_negotiation.html

Browsers and HTTP

Throughout the tutorial the use of clients and agents is noted, often avoiding the mention of Web browsers. The reason is that there is more than one way to use HTTP to connect to a Web site. In fact, nearly anything done in a Web browser can be done from the command line.

View the Web page "I sit in Siberia" for more information on how to use the CLI in conjunction with HTTP to access Web sites (http://www.pfuca.com/products/products.html), or, from a Telnet window, type the following:

```
www.pfuca.com 80
```

Press Enter, then type:

```
GET /products/products.html HTTP/1.0
```

Or, the following may also be typed:

```
GET /products/products.html HTTP/1.1
Host: stud1.tuwien.ac.at
```

Then press Enter twice and wait for output, which will be something like the following:

```
Bash$ wget yahoo.com
--15:29:43--  http://yahoo.com:80/
           => 'index.html'
Connecting to yahoo.com:80... connected!
HTTP request sent, awaiting response... 302 RD
Location: http://www.yahoo.com/ [following]
--15:29:44--  http://www.yahoo.com:80/
           => 'index.html'
Connecting to www.yahoo.com:80... connected!
HTTP request sent, awaiting response... 200 OK
Length: 16,512 [text/html]
0K -> .......... ......                                          [100%]
15:29:45 (143.97 KB/s) - 'index.html' saved [16512/16512]
```

Style Sheets

Web developers use style sheets to control aspects of a Web site's style (such as fonts, background colors, or table attributes) from a central place.

Even though there has been a standard since at least 1997, style sheets are still a rare thing. This is attributed to the fact that multiple versions of Web browsers on various platforms have different levels of support for the technology. Figures 1.2 through 1.4 are examples of style sheets for various applications.

Despite the apparent attempts of the main browser makers to degrade the style sheet standard, style sheets are useful. Additionally, dedicated developers use workarounds to get each browser to display content in a similar way.

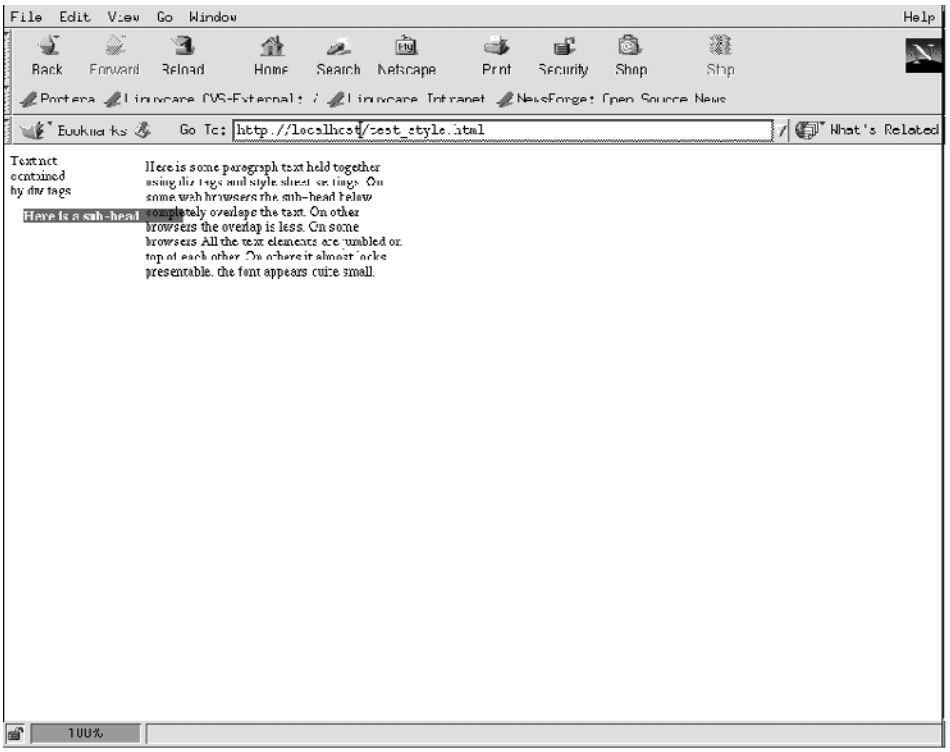


Figure 1.2 Screenshot of a Web page using the style sheet displayed in Netscape 4.7.3 for Linux.

Java

One can use Java to create dynamic content for Web pages. There are two ways to go about it. One is to embed a premade Java applet in a Web page, and another is to write a custom Java application. Java is likened to C++ in its complexity, which is daunting to many developers.

While writing Java can be difficult for those not familiar with object-oriented programming, embedding Java applets is something most Web developers can master easily.

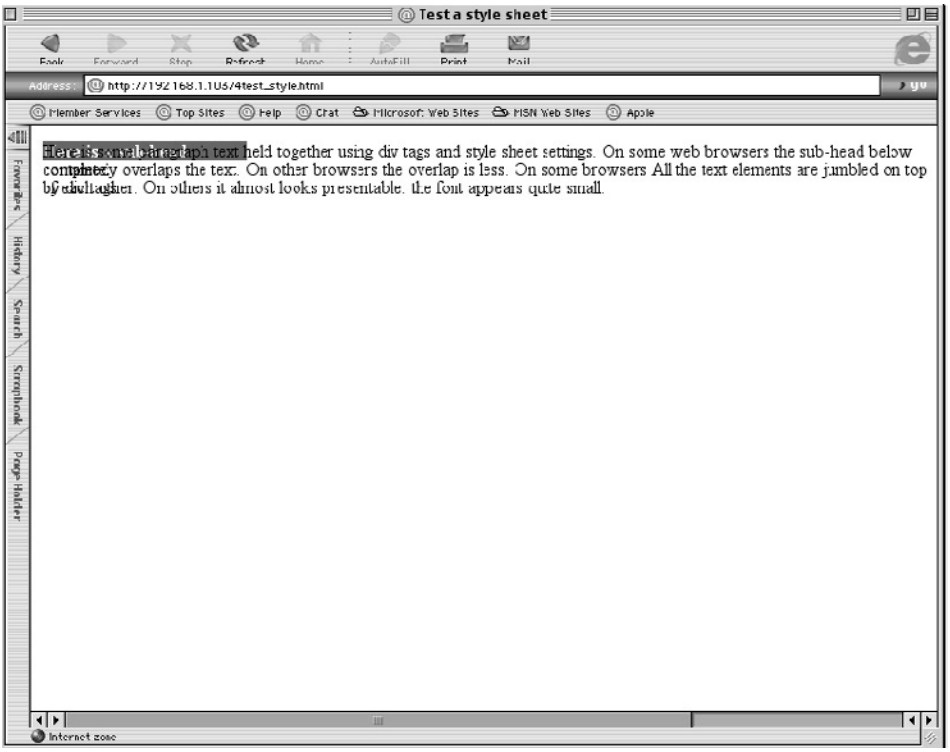


Figure 1.3 Same Web page/style sheet combination from Figure 1.2 rendered by Internet Explorer 5.0 on a Macintosh.

Generally the code for embedding an applet looks something like the following:

```
<applet codebase="../LED" code="LED.class" width=500 height=48 align=center>
   <param name="script" value="../scripts/tutorial.led">
   <param name="border" value="2">
   <param name="bordercolor" value="100,130,130">
   <param name="spacewidth" value="43">
   <param name="wth" value="122">
   <param name="ht" value="9">
   <param name="font" value="../fonts/default.font">
   <param name="ledsize" value="3">
   <hr>
   <BR>
   <IMG SRC="LEDSign.gif">
   <hr>
</applet>
```

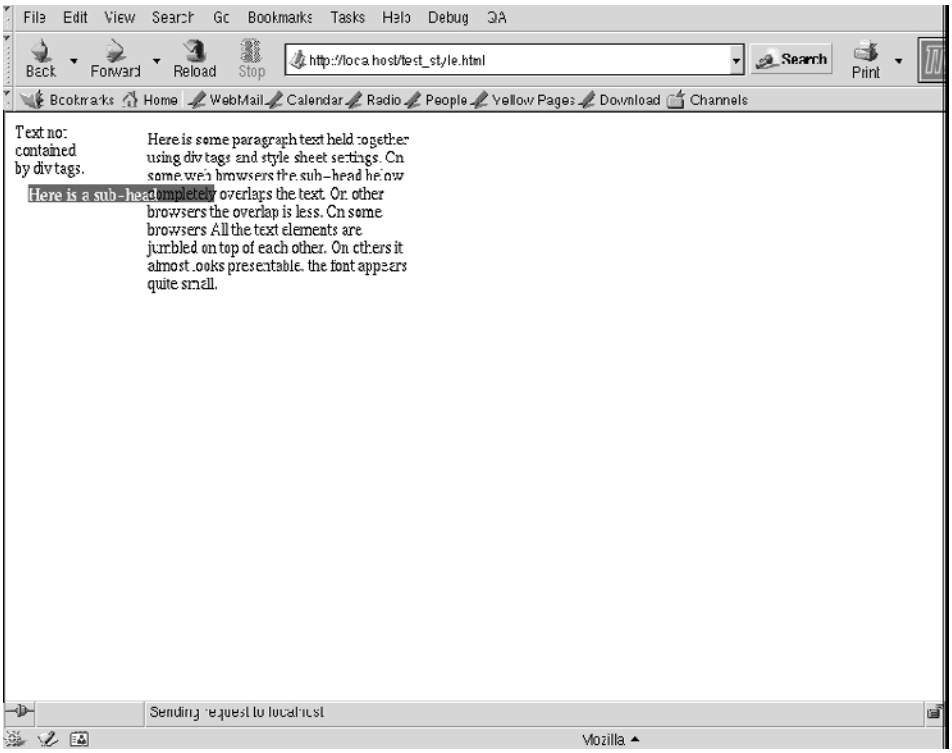


Figure 1.4 Screenshot of style sheet output in Mozilla 0.8 for Linux.

A Java-enabled Web browser would display a scrolling LED sign.

JavaScript

JavaScript and Java are unrelated except for some object-oriented principles and similar sounding names. JavaScript is compiled by the Web browser with the rest of an `html` file. Java is served compiled already, just like any binary file. JavaScript code is generally embedded directly in an html page. Java applets are called using the `<applet>` tag and can exist anywhere on a server or network.

Like Java, JavaScript serves dynamic content. Unlike Java, content is read and compiled by the browser. This should come to your attention immediately if you read the section on style sheets; multiple versions of

various browsers on various platforms can behave differently when they encounter different versions of JavaScript. Most browsers try to be backward compatible with older versions of scripting language, but that compatibility can be difficult.

Base Systems

Running a multiserver setup on any machine requires a more advanced base system than running a single application machine (see Figure 1.5). Each base system must be able to support the demands of all possibly running servers. The following sections will describe the details of the base system requirements needed by Apache. This will give administrators an idea of the factors to consider when adding Apache.

Preparing Linux

Before any server is created, one should optimize the hardware for that server. The following sections outline general steps that can be taken to prepare a Linux machine before any installation begins.

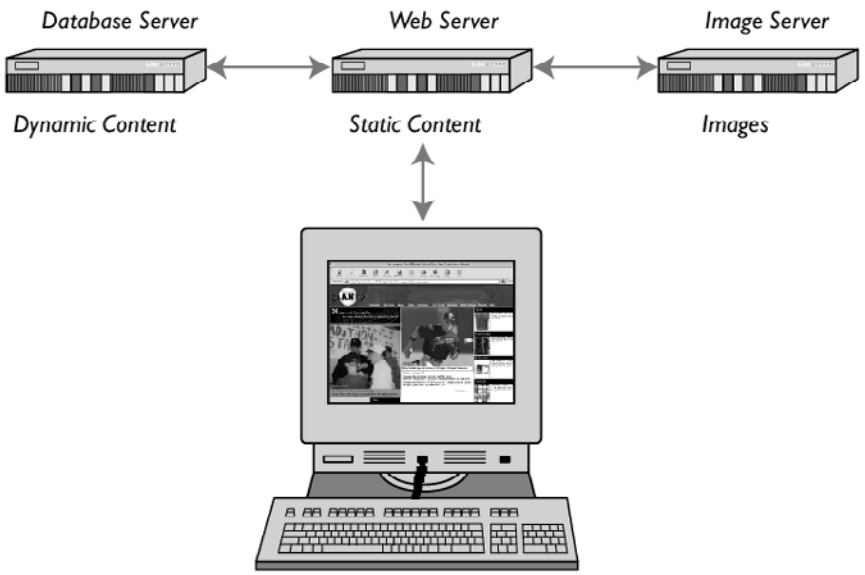


Figure 1.5 Typical multiple server setup preparing Linux.

RAM

System memory is by far the most necessary resource for a server that handles a high volume of Web requests. Once the physical memory has been exhausted, the server machine will begin to utilize its swap memory, which will slow it down. Frequently monitor the memory usage of the server. If a large amount of swap space is used, increase the amount of RAM on the machine.

If possible, go to the top of the directory that will serve all of the Web pages, and use the `du -s` command to see how much RAM it would take to hold them all. There must be enough memory to hold all of the Web pages plus an extra 50MB to hold the operating system and the applications that are running.

Hardware

For high loads, faster is better. For reference, an old Pentium-based machine with adequate memory can use most of a 10MB connection. This varies widely, depending on factors such as the type of content and network conditions. Hardware has become much faster since the original Pentium, so a modest machine by modern standards should still be usable for many applications.

Again, monitor the machine's performance. Use `top` or another program to determine the load on the server. If the load average is frequently high, consider upgrading to a faster system. The following shows the output of `top`:

```
11:56am  up 11 days,  3:52,  1 user,  load average: 0.67, 0.21, 0.07
115 processes: 112 sleeping, 3 running, 0 zombie, 0 stopped
CPU states: 45.7% user, 54.2% system,  0.0% nice,  0.0% idle
Mem:   196000K av,  173480K used,   22520K free,       0K shrd,   4948K buff
Swap:  249440K av,   28456K used,  220984K free        51196K cached

  PID USER     PRI  NI  SIZE  RSS SHARE STAT %CPU %MEM   TIME COMMAND
 6247 apache    11   0  4580 4504  4288 S     9.6  2.2   0:00 httpd
 6251 apache    11   0  4580 4504  4288 S     9.6  2.2   0:00 httpd
 6249 apache    11   0  4580 4504  4288 S     9.0  2.2   0:00 httpd
 6252 apache    11   0  4580 4504  4288 S     9.0  2.2   0:00 httpd
 6248 apache    11   0  4580 4504  4288 S     8.5  2.2   0:00 httpd
 6250 apache    11   0  4580 4504  4288 S     8.5  2.2   0:00 httpd
 6253 apache    11   0  4580 4504  4288 S     6.2  2.2   0:00 httpd
 6254 apache    11   0  4580 4504  4288 S     3.4  2.2   0:00 httpd
```

```
6255 apache    11   0  4580 4504  4288 S     1.7  2.2   0:00 httpd
6257 apache    11   0  4580 4504  4288 S     1.7  2.2   0:00 httpd
6259 apache    11   0  4580 4504  4288 S     1.7  2.2   0:00 httpd
6256 apache    11   0  4580 4504  4288 S     1.1  2.2   0:00 httpd
6258 apache    11   0  4580 4504  4288 S     1.1  2.2   0:00 httpd
6260 apache    11   0  4584 4508  4288 R     1.1  2.3   0:00 httpd
```

Hard Drive

Loading and running a Web site from RAM offers the fastest delivery of Web pages. Unfortunately, this configuration may not be possible; therefore, the hard drive must be accessed. Accessing the hard drive is significantly slower than RAM. To offset this performance lapse, use faster hard drives.

SCSI drives are generally faster than IDE drives because they put less of a load on the system. If a Web server is already taxing the CPU, it is a good idea to remain with the SCSI drive.

To optimize hard drive performance, take the following into account:

Swap space. People often forget to put swap space on a different partition from the rest of the data. Not doing this can greatly slow down the Web server. If possible, put it on a completely different disk. Putting the swap partition on the first partition of the disk can actually increase transfer rates for some data by a factor of 2.

Multiple drives and logs. Apache logs every request that hits it. All this logging means much writing to disk. If multiple drives are available, put the Apache logs (as well as any other logs) on the fastest drive of the bunch.

Logging. If you are not planning to use logging, turn it off. While doing this, the speed enhancement will be substantial. This could be dangerous, though, because Apache will not write every request and error to a disk. If logging is desired, define the levels that work best with the system. There will be a difference in the error log when LogLevel is changed from debug to emerg.

Kernel

If performance is a major issue, do not run a stock kernel. Linux distributions tend to have a large kernel to accommodate the variety of potential hardware types that might be used with it. As a consequence, it can be slower than a kernel that has been configured by hand and pared down.

For maximum performance, recompile the kernel with the bare minimum that is required for the machine's needs. It will be more efficient and will also allow more of it to fit in the CPU cache. Use `/sbin/lsmod` to see what modules have been loaded with the stock install. The following shows a list of loaded modules:

```
# /sbin/lsmod
Module                  Size  Used by
nls_cp437               3952   1 (autoclean)
soundcore               2800   0 (autoclean) (unused)
lockd                  32208   1 (autoclean)
sunrpc                 54640   1 (autoclean) [lockd]
irda                   80304   1
autofs                  9456   2 (autoclean)
usb-uhci               19184   0 (unused)
usbcore                43632   1 [usb-uhci]
serial_cs               5456   0 (unused)
3c574_cs               10240   1
ds                      6448   2 [serial_cs 3c574_cs]
i82365                 22928   2
pcmcia_core            45984   0 [serial_cs 3c574_cs ds i82365]
```

The Linux kernel is always changing. There are some changes with every release, including the TCP/IP stack, which is important for Apache. It is an especially good idea to watch for changes in the networking code because they may be beneficial to users who want more performance out of their Web servers. Newer versions of Apache may be better optimized as well. For production systems, wait a few weeks before using a new version to ensure that it is stable.

File Handles and Inodes

Each file that is opened on the system must have an associated file handle, or inode, that the kernel uses to keep track of it. The latest Linux kernels have a default limit of 4,096 file handles. If the server will have many files open at once, the limit may be exceeded. This is important when running multiple virtual hosts with separate log files.

Introduction to Packages

Linux software is generally distributed in a format called a package. A package is a collection of files combined into a single file to simplify distribution and installation. A good package system will also contain dependencies. A package can list all of the prerequisite packages that must be

installed before it can run. The role of a system administrator is to maintain packages by installing, upgrading, removing, and verifying them.

The three most common packaging formats are RPMs, DEBs, and tarballs. RPMs are used in Red Hat, Caldera, and most other distributions. DEBs are used in Debian and Corel Linux. Tarballs are used by Slackware and in source packages.

RPM Security Tasks

The following is a list of suggestions for security procedures for software that has been acquired from the Internet. To maintain a high level of security, practice these on a regular basis.

Issues with Stock Vendor Builds

If the user installs the packaged Apache that comes with the distribution, he or she should rely heavily on the documentation that comes with the package. Paths, server defaults, module options, and any number of things follow a style that is unique and limited to that particular distribution.

If your location has specific needs with respect to modules or will rely on Apache for many aspects, obtain the sources and build it yourself. Keep in mind that the prepackaged distributions are built in a way that will accommodate a varying range of needs. If your needs stray outside of this range, you will have to do this work anyway. This process will be easier the earlier it is started.

Security Updates

Prepackaged distributions are designed to run on most systems. They include many services that the user and the system administrator need to either update or remove from the system. For example, the "Ramen worm" currently infects only Red Hat servers that have not had all of the security updates applied, as per the CNETNews.com story dated January 17, 2001 (http://news.cnet.com//news/0-1003-200-4508359.html).

Validating Packages

Verifying a package compares information about files with the same information from the original package. Verifying compares the size, MD5 sum, permissions, type, owner, and group of each file.

The command `rpm -V` verifies a package.

For example, to verify a package containing a particular file:

```
rpm -Vf /bin/vi
```

To verify all installed packages:

```
rpm -Va
```

To verify an installed package against an RPM package file:

```
rpm -Vp foo-1.0-1.i386.rpm
```

This verification can be useful if the user suspects that the RPM databases are corrupt.

If everything is verified properly, there will be no output. Any discrepancies will be displayed. The format of the output is a string of eight characters, a possible `c` denoting a configuration file, and then the file name. Each of the eight characters denotes the result of a comparison of one attribute of the file to the value of that attribute recorded in the RPM database.

Test Results

A single period (.) means the test passed. The following characters denote failure of certain tests:

`5`	MD5 checksum.
`S`	File size.
`L`	Symbolic link.
`T`	File modification time.
`D`	Device.
`U`	User.
`G`	Group.
`M`	Mode (includes permissions and file type).

Verify the Update Prior to Use

To ensure the integrity of the downloaded package, use this command:

```
rpm --checksig package.rpm
```

RPM Installation

Installing an RPM-based distribution is a matter of invoking `rpm` to extract and install the contents:

```
# rpm -i apache-x.x.x.rpm
```

When the install finishes, Apache should be ready to run.

Starting, configuring, and maintaining are roughly the same as with a source install. One caveat with using an Apache RPM is that there are many Linux distributions that implement RPM for managing packages but little consensus as to where certain files belong. If the user cannot find a central Apache or `http` directory and the install seems to have completed properly, the distribution probably has spread the Apache files across several directories. Use this command to list all installed files:

```
# rpm -ql apache | less
```

The user should be able to find the configuration and content directories in this list.

Some of the important files are placed here using the Apache package included with Red Hat 6.2:

```
httpd.conf: /etc/httpd/conf
HTML content: /home/httpd/html
httpd binary: /usr/sbin
```

NOTE **apachectl is not included with this RPM, but there is a startup script in /etc/rc.d/ init.d called httpd.**

The mileage may vary depending on the distribution. The source is always available if an RPM install does not meet the needs of your location. A source installation also provides the user with the opportunity to select the features that he or she wishes to enable.

NOTE **Apache may be split into multiple packages in some distributions. For example, Red Hat has split out some of the files into a package called apache-devel. This package contains files required to build third-party modules.**

Locate Security Updates

To locate the latest versions of a vendor's security updates, go to the vendor's home page or one of its mirrors and type the following search string in the `Search` box:

```
RH 6.2 + security
```

The following is an abridged screen dump of the security updates available for t 6.2 as of January 2001:

```
From - Sun Jan  7 22:42:06 2001
ftp://rpmfind.net/linux/redhat/updates/6.2/i386/
Directory of /linux/redhat/updates/6.2/i386
Current directory is /linux/redhat/updates/6.2/i386
Up to higher level directory
SysVinit-2.78-5.i386.rpm                          80 Kb    Thu Nov  9 20:45:00 2000
apache-1.3.14-2.6.2.i386.rpm                      420 Kb   Mon Oct 23 20:35:00 2000
apache-devel-1.3.14-2.6.2.i386.rpm                108 Kb   Mon Oct 23 20:36:00 2000
apache-manual-1.3.14-2.6.2.i386.rpm               462 Kb   Mon Oct 23 20:36:00 2000
auth_ldap-1.4.0-3.i386.rpm                        26 Kb    Mon Oct 23 20:36:00 2000
imap-2000-2.6.i386.rpm                            1034 Kb  Fri Nov 10 16:45:00 2000
imap-devel-2000-2.6.i386.rpm                      1118 Kb  Fri Nov 10 16:48:00 2000
 kernel-2.2.16-3.i386.rpm                         5818 Kb  Mon Jun 26 00:00:00 2000
 kernel-BOOT-2.2.16-3.i386.rpm                    3351 Kb  Mon Jun 26 00:00:00 2000
 kernel-doc-2.2.16-3.i386.rpm                     953 Kb   Mon Jun 26 00:00:00 2000
 kernel-headers-2.2.16-3.i386.rpm                 1128 Kb  Mon Jun 26 00:00:00 2000
mod_perl-1.23-3.i386.rpm                          746 Kb   Mon Oct 23 20:37:00 2000
netscape-common-4.76-0.6.2.i386.rpm               8326 Kb  Fri Nov 17 19:50:00 2000
netscape-communicator-4.76-0.6.2.i386.rpm         5909 Kb  Fri Nov 17 19:51:00 2000
netscape-navigator-4.76-0.6.2.i386.rpm            3143 Kb  Fri Nov 17 19:45:00 2000
nss_ldap-122-1.6.i386.rpm                         96 Kb    Fri Oct 27 19:05:00 2000
openldap-1.2.9-6.i386.rpm                         1326 Kb  Fri Nov 10 17:00:00 2000
openldap-devel-1.2.9-6.i386.rpm                   128 Kb   Fri Nov 10 17:00:00 2000
openssl-0.9.5a-2.6.x.i386.rpm                     1170 Kb  Fri Nov 10 16:46:00 2000
openssl-devel-0.9.5a-2.6.x.i386.rpm               894 Kb   Fri Nov 10 16:46:00 2000
openssl-perl-0.9.5a-2.6.x.i386.rpm                8 Kb     Fri Nov 10 16:46:00 2000
openssl-python-0.9.5a-2.6.x.i386.rpm              123 Kb   Fri Nov 10 16:46:00 2000
perl-5.00503-12.i386.rpm                          4923 Kb  Fri Aug 18 22:45:00 2000
php-3.0.17-1.6.2.i386.rp                          306 Kb   Mon Oct 23 20:37:00 2000
php-imap-3.0.17-1.6.2.i386.rpm                    352 Kb   Mon Oct 23 20:35:00 2000
php-ldap-3.0.17-1.6.2.i386.rpm                    14 Kb    Mon Oct 23 20:36:00 2000
php-manual-3.0.17-1.6.2.i386.rpm                  968 Kb   Mon Oct 23 20:37:00 2000
php-pgsql-3.0.17-1.6.2.i386.rpm                   16 Kb    Mon Oct 23 20:36:00 2000
python-1.5.2-27.6.x.i386.rpm                      1805 Kb  Thu Dec 21 21:32:00 2000
python-xmlrpc-1.2.1-0.6.x.i386.rpm                35 Kb    Thu Dec 21 21:32:00 2000
```

Add Modules via RPMs

Adding new modules to an RPM-based distribution is a matter of invoking RPM to upgrade/install the new module:

```
# rpm -Uvh apache-x.x.x.rpm
```

When the upgrade finishes, Apache should be ready to run.

Start Apache

Red Hat uses `httpd` to start the server.

httpd

`httpd` is a script that can start Apache's daemon (located in `/usr/sbin`). The script is located at `/etc/rc.d/init.d/httpd`. To see the command line options, type:

```
Usage: ./httpd {start|stop|restart|reload|status}
```

When the user types `httpd -- help`, the `httpd` command-line options will be visible, as shown in Table 1.1.

Verify Functionality Locally

The final step of upgrading Apache involves testing the server. When the daemon is running, Apache should be listening on the loopback device at `port 80`. Typically the loopback device resides on 127.0.0.1 and can be accessed using `localhost` as the target address. Any Web browser can perform this test. Lynx is a good choice because it can usually be found on any Linux distribution. Test the server by invoking Lynx or a preferred browser with the desired server address:

```
# lynx localhost
```

The Apache test page should load with the following screen:

```
It Worked! The Apache Web Server is Installed on this Web Site!
If you can see this page, then the people who own this domain have just installed the
Apache Web server software successfully. They now have to add content to this directory
and replace this placeholder page, or else point the server at their real content.
```

Table 1.1 Options for httpd Help

`-D name`	Define a name for use in <IfDefine name> directives
`-d directive`	Specify an alternate initial ServerRoot
`-f file`	Specify an alternate ServerConfigFile
`-C Ðdirective"`	Process directive before reading config files
`-c Ðdirective"`	Process directive after reading config files
`-v`	Show version number
`-V`	Show compile settings
`-h`	List available command-line options
`-l`	List compiled-in modules
`-L`	List available configuration directives
`-S`	Show parsed settings (currently only vhost settings)
`-t`	Run syntax check for config files (with docroot check)
`-T`	Run syntax check for config files (without docroot check)

```
If you are seeing this page instead of the site you expected, please contact the
administrator of the site involved. Although this site is running the Apache software
it almost certainly has no other connection to the Apache Group, so please do not
send mail about this site or its contents to the Apache authors. If you do, your
message will be ignored.
The Apache documentation has been included with this distribution.
The Webmaster of this site is free to use the image below on an Apache-powered Web
server.
Thanks for using Apache!
```

If there is an error, verify that the command was entered correctly and try again. If the test page still does not load, return to the installation section and perform the configuration and compilation steps again.

Verify Functionality over a Network

Once Apache is functioning correctly on the local system, the user can perform a network test. If there are nearby machines on a local network, the user can go to any of those machines and use a browser. Access the server with the IP address of the machine's network interface to eliminate any problems that DNS may cause.

A listing of all available network interfaces on the system can be found by invoking `ifconfig` without any options:

```
# ifconfig
eth0  Link encap:Ethernet HWaddr 00:80:C8:14:73:77
      inet addr:10.168.1.2  Mask:255.255.255.0
      UP BROADCAST RUNNING MTU:1500  Metric:1
      RX packets:20477 errors:0 dropped:0
      TX packets:20895 errors:0 dropped:0
      collisions:0 txqueuelen:100
      Interrupt:5 Base address:0x340
lo    Link encap:Local Loopback
      inet addr:127.0.0.1  Mask:255.0.0.0
      UP LOOPBACK RUNNING  MTU:3924  Metric:1
      RX packets:3993 errors:0 dropped:0
      TX packets:3993 errors:0 dropped:0
      collisions:0 txqueuelen:0
```

The `inet addr:10.168.1.2` portion indicates the local IP address of the system.

Lynx or any other browser can be used for this test:

```
# lynx 10.168.1.2
```

(Use your actual IP address here.)

If all is configured properly, the Apache test page should load in the browser. If the user does not have a browser, use telnet to connect.

Next, test the page using a name instead of an IP address.

Two Ways to Map a Domain Name to an IP Address

The simplest way is to put an entry in the `/etc/hosts` file of the client. The entry will have a format like this:

```
10.168.1.1 alias1 alias2.domain.com alias3.xyz.cx
```

The format is the IP address followed by all of the names that map to it.

The alternative is to use a DNS. DNS is beyond the scope of this course, but if a server has already been added to an IP network, adding a DNS entry should not be a problem.

Source RPMs

Software development using RPM proceeds in strict stages from sources with local patches applied, through configuration, compilation, and installation. The final output of a build typically consists of a source package and one or more binary packages that can be installed. The entire build process is described in something called a spec file.

The focus on encapsulating all components necessary to produce a binary package in RPM prevents reuse of common code. The encapsulation is called a source RPM and consists of the sources, local patches, and the spec file describing the build process. The spec file is augmented with certain configuration information such as the compiler flags to use and the architecture on which the package is being built. The other information is of little use in preparing a package.

The example that follows describes the structure of SRPMS. It assumes the user has a Red Hat CD-ROM RPM for the Apache program:

```
# ls
COPYING README SRPMS TRANS.TBL
# cd SRPMS
# ls apache*
apache-x.x.x.x.src.rpm
# rpm -i apache-x.x.x.x.src.rpm
# cd /usr/src/redhat/SPECS
# ls
apache.spec
# ls /usr/src/redhat/SOURCES/
```

The `rpm -i apache*` command installed the patches in the `/usr/src/redhat/SOURCES` directory. The specs file describes the architecture-specific compilation procedure.

Installation

Before unpacking a tarball, examine the contents. To do this, run the following command:

```
tar -tzvf apache_1.3.19.tar.gz
```

To unpack a tarball, first change to the directory where it will be unpacked. Then, execute the following command:

```
tar -xzvf apache_1.3.19.tar.gz
```

This will usually create a subdirectory and expand the package into it.

Compiling

To compile a program from the source, first change to the directory where the package was unpacked. This is called the top-level directory of the package. It is usually named something similar to `apache_1.3.14`.

```
:/usr/src$ cd apache_1.3.19
```

After unpacking the tarball and entering the resulting subdirectory, perform a step that is common to most source code installations, a screen dump of `./configure --help`.

```
:/usr/src/apache_1.3.14$ ./configure --help | less

--activate-module=scr/modules/auth_mysql/libauth_myslq.a
 auth_mysql modules.
```

Add the following line to the Configuration file before compiling the server:

```
AddModule modules/standard/mod_rewrite.o
--activate-module=scr/modules/auth_mysql/libauth_myslq.a
# ./configure --enable-module=so --enable-rule=EAPI

mod_status, mod_update
Usage: configure [options]
Options: [defaults in brackets after descriptions]
General options:
 --quiet, --silent      do not print messages
 --verbose, -v          print even more messages
 --shadow[=DIR]         switch to a shadow tree (under DIR) for building

Stand-alone options:
 --help, -h             print this message
 --show-layout          print installation path layout (check and debug)

Installation layout options:
 --with-layout=[F:]ID   use installation path layout ID (from file F)
 --target=TARGET        install name-associated files using basename TARGET
 --prefix=PREFIX        install architecture-independent files in PREFIX
 --exec-prefix=EPREFIX  install architecture-dependent files in EPREFIX
 --bindir=DIR           install user     executables in DIR
 --sbindir=DIR          install sysadmin executables in DIR
 --libexecdir=DIR       install program  executables in DIR
```

```
 --mandir=DIR            install manual pages in DIR
 --sysconfdir=DIR        install configuration files in DIR
 --datadir=DIR           install read-only data files in DIR
 --iconsdir=DIR          install read-only icon files in DIR
 --htdocsdir=DIR         install read-only document files in DIR
 --cgidir=DIR            install read-only cgi files in DIR
 --includedir=DIR        install includes files in DIR
 --localstatedir=DIR     install modifiable data files in DIR
 --runtimedir=DIR        install runtime data in DIR
 --logfiledir=DIR        install logfile data in DIR
 --proxycachedir=DIR     install proxy cache data in DIR

Configuration options:
 --enable-rule=NAME      enable  a particular Rule named ÔNAME'
 --disable-rule=NAME     disable a particular Rule named ÔNAME'
                         [DEV_RANDOM=default EXPAT=default   IRIXN32=yes ]
                         [IRIXNIS=no      PARANOID=no     SHARED_CHAIN=de]
                         [SHARED_CORE=default SOCKS4=no       SOCKS5=no  ]
                         [WANTHSREGEX=default                            ]
 --add-module=FILE       on-the-fly copy & activate a 3rd-party Module
 --activate-module=FILE  on-the-fly activate existing 3rd-party Module
 --permute-module=N1:N2  on-the-fly permute module ÔN1' with module ÔN2'
 --enable-module=NAME    enable  a particular Module named ÔNAME'
 --disable-module=NAME   disable a particular Module named ÔNAME'
                         [access=yes      actions=yes     alias=yes      ]
                         [asis=yes        auth_anon=no    auth_dbm=no    ]
                         [auth_db=no      auth_digest=no  auth=yes       ]
                         [autoindex=yes   cern_meta=no    cgi=yes        ]
                         [digest=no       dir=yes         env=yes        ]
                         [example=no      expires=no      headers=no     ]
                         [imap=yes        include=yes     info=no        ]
                         [log_agent=no    log_config=yes  log_referer=no ]
                         [mime_magic=no   mime=yes        mmap_static=no ]
                         [negotiation=yes proxy=no        rewrite=no     ]
                         [setenvif=yes    so=no           speling=no     ]
                         [status=yes      unique_id=no    userdir=yes    ]
                         [usertrack=no    vhost_alias=no                 ]
 --enable-shared=NAME    enable  build of Module named ÔNAME' as a DSO
 --disable-shared=NAME   disable build of Module named ÔNAME' as a DSO
 --with-perl=FILE        path to the optional Perl interpreter
 --with-port=PORT        set the port number for httpd.conf
 --without-support       disable the build and installation of support tools
 --without-confadjust    disable the user/situation adjustments in config
 --without-execstrip     disable the stripping of executables on installation
 --server-uid=UID        set the user ID the web server should run as [nobody]
 --server-gid=GID        set the group ID the web server UID is a member of [#-1]

suEXEC options:
 --enable-suexec         enable the suEXEC feature
 --suexec-caller=NAME    set the suEXEC username of the allowed caller [www]
 --suexec-docroot=DIR    set the suEXEC root directory [PREFIX/share/htdocs]
```

```
 --suexec-logfile=FILE  set the suEXEC logfile [PREFIX/var/log/suexec_log]
 --suexec-userdir=DIR   set the suEXEC user subdirectory [public_html]
 --suexec-uidmin=UID    set the suEXEC minimal allowed UID [100]
 --suexec-gidmin=GID    set the suEXEC minimal allowed GID [100]
 --suexec-safepath=PATH set the suEXEC safe PATH [/usr/local/bin:/usr/bin:/bin]
 --suexec-umask=UMASK   set the umask for the suEXEC'd script [server's umask]

Deprecated options:
 --layout               backward compat only: use --show-layout
 --compat               backward compat only: use --with-layout=Apache
```

Most packages that a user wants to compile will probably contain adequate documentation that explains how to configure and build the software. The first step is to read through that information. The primary place to search for that information is in the `README` file that came with the source code. This will often provide a quick overview of the program and either will tell how to compile it or will list where to look to find that information.

Most GNU software and recently released free software use the `autoconf` system. This means that the configure script can be run to let the software determine what features are available on your system.

This will list all of the options to the configure script. The ones at the top are standard options that are generally used to tell the script where to install all of the files. The most important of these is the `--prefix` option. It usually defaults to the `/usr/local` directory.

To configure a package that installs the `/opt/program` directory instead, run configure with the following command:

```
./configure --prefix=/opt/program
```

The options near the bottom of the help screen usually differ from package to package. Read the included documentation to determine which features should be enabled. The defaults are sufficient in most cases.

As the list is quite extensive, we will discuss only the options we are using and recommend. The `--with-layout` option allows specification of the directory structure for installation. These layouts are the same as those provided by the different distribution packages. For more information, check the `config.layout` file. Users can also add their own layouts to the `config.layout` or pass the appropriate configuration options during the `./configure` step.

Binary distributions usually include all options as shared object modules, probably to provide choices for users. The shared object modules that come with Apache by default are usually desirable. Therefore, compile them and allow for additional modules to be loaded as needed. The default is to compile all the default modules.

–with-layout=RedHat

This will configure your Apache files to conform to Red Hat's file layout. The following is a `./configure` line:

```
:/usr/src/apache_1.3.14$ ./configure \
--with-layout=RedHat --enable-module=so
```

Dynamic Shared Objects

The `--enable-module=so` option gives the ability to add other modules at a later time or at run time. These are analogous to loadable modules and the Linux kernel. For example, to add php to the server, simply add `mod_php` without recompiling Apache.

Once the configure script has run and determined the features available on a system, the software can be built.

Installing

Once the source is compiled into an executable, it can be installed. To install a program, write access is required for the directories in which the files will be installed. In most cases, this means logging in to the root account.

NOTE **Only the installation phase requires root access. The compilation phase does not require root access, so it should be done as a nonprivileged user to prevent problems. If you do not have root access, it may be possible to install the package into a directory in which you have write access. This is generally specified using the `--prefix` option in the configuration phase.**

Testing before Installing

On many packages, there is a command to test the program and ensure that it compiled correctly. This command takes one of two forms:

```
:/usr/src/apache_1.3.14$ make check
:/usr/src/apache_1.3.14$ make test
```

If one of these is available, it will run a series of tests and report whether the program will run correctly. It may be possible to run the program to test it before it is installed. Be sure to specify that the program should run from the current directory:

```
:/usr/src/apache_1.3.14$./program
```

Many programs cannot run until installed. Assuming there are no errors, continue with the installation:

```
:/usr/src/apache_1.3.14$ make
:/usr/src/apache_1.3.14$ make install
```

This will copy all of the programs and all of the associated files to their proper locations. Once the installation process is complete, log out of the root account and try the program.

This process will often take quite a while, depending on the size of the package and the memory and speed of the computer. When the process is complete, the user should be returned to the command prompt with no error messages displayed. If there is an error, the cause must be determined. The most common cause is a missing package.

System Utilities

System utilities ease the installation and maintenance of Apache. They also shorten update times and narrow the margin of installation error. Utilities are normally used for average Web server installations, thus eliminating source downloads and source compilations. Most users will use utilities to update and maintain software because utilities provide a basis of operation, maintenance, and configuration for the Apache Web server.

The httpd Daemon

The `httpd` daemon is used by Web servers to talk to clients (such as Netscape) via the HTTP protocol. The `httpd` daemon is usually run on port 80. It offers users many options for Apache configuration and customization.

Defaults

Use the defaults if running a single daemon; there is no special content that must be provided. Making no changes prior to compiling should result in

a basic Web server that is ready immediately after compiling. Some of the defaults are covered in this section.

Apache's home directory will be `/usr/local/apache` for Red Hat. (Default in Debian is `/etc/apache` and `/var/lib/apache` in Slackware.) All supporting files and directories will be based from this location.

The server will listen on all network interfaces that exist on the host. This may or may not be desired, but for a default install it ensures that Apache will be listening on an active interface and be accessible for testing.

Paths, startup configurations, and a test page are all provided.

Configuring with defaults is just a matter of invoking the following command in the source directory:

```
$ ./configure
```

This is fairly simple. A lot of work has gone into providing this clean and easy installation process, available since the release of Apache 1.3.

Customization

As mentioned previously, Apache uses something similar to the GNU configure script to prepare everything for compiling. One aspect of Apache's script is a wide variety of options that `configure` can take as arguments. For a general install, there is no need to learn them all.

Another important configuration option involves steps to include modules, which will be discussed briefly to prepare for later chapters where they will be needed for things like CGI scripting.

It is often desirable to have additional functionality with a Web server, such as scripting or database integration. Additionally, an administrator may not want to have Apache in the default directory of `/usr/local/apache`. For specifying changes to the default, they can be added at the command line as options to `configure`. Invoking configure with command-line options will generally take the following form:

```
$ ./configure --some_option=argument_forthatoption
]
```

so that a real-world option is similar to the following:

```
$ ./configure --prefix=/www
```

The `prefix` option tells the `configure` scripts where the server will be installed on the system. Looking at the preceding example, we see that Apache will have all of its files reside within the `/www` directory. The arguments to `--prefix` are free form, so to use `--prefix=/missouri`, the install scripts would use `missouri` for Apache's home directory.

Another option is the `--with-layout=GNU` option. This will cause files to be installed in locations compliant with the GNU standards. This is how most distributions install their Apache packages.

Modules and SSL

This module uses a rule-based rewriting engine to rewrite requested URLs. This feature exists in Apache 1.2 and later. With its powerful URL manipulation mechanism, the Web server can examine every URL that arrives to see if it matches any patterns specified by the rewrite rules. If it makes a match, Apache internally rewrites a URL using that rule. Here the module operates on full URLs in both a per-server context (`httpd.conf`) and a per-directory context (`htaccess`). To include `mod_rewrite`, add the following line to the `Configuration` file before compiling the server:

```
AddModule modules/standard/mod_rewrite.o
```

Another use of `mod_rewrite` is with proxy servers to make sure people can use the proxy only if they come from a given URL, in this case `.linux.org`:

```
RewriteRule !^proxy:http://[^/]*\.linux\.org/ -  \ [forbidden]
```

The Apache 1.3 HTTP server is now successfully built and installed. To verify that Apache actually works correctly, first check the initially created or preserved configuration files `/etc/httpd/conf/ httpd.conf`; then Apache can be started for the first time by running:

```
Bash$ /usr/sbin/apachectl start
      Thanks for using Apache.
      The Apache Group
      http://www.apache.org/ ********************
```

Module Performance and Functionality

Apache implements a high-performance Web server with high yields in stability. The combination of Apache and Linux provides the modular design, speed, and stability that is essential for a great Web server. With its modular architecture, users may either add or remove modules to adapt its functionality for individual needs. Modules make Apache the best choice for a Web server in a scalable environment. A wide variety of modules is available to cover most needs.

Users can compile Apache with dynamic modules and add or remove them as needed. For example, to increase security, search the Internet and download `mod_ssl` and turn an existing Apache server into a secure server with 128-bit encryption.

Apache is the most flexible and extensible Web server available today. Internet polls have shown it to be the most prevalent server as well.

On top of Apache's SSL capabilities and its modular design, users can also run Common Gateway Interface (CGI) programs. CGI is a standard that allows programs to communicate with the Web server, as well as transfer information to and from the World Wide Web. Any program can run as a CGI as long as Apache is configured properly. CGI allows the program to communicate with the Web browser and run the user as a protected process. Under Linux, CGI programs run the user as Apache. This allows users to forget about some of the security concerns associated with the scripts running as root. CGI scripts are a very important part of the Apache Web server; however, Apache's resources are not limited to SSL, CGI, and modules. Apache also provides great documentation on all of its features available on its Web site.

Enabling Modules

The Apache server implements modules to enhance the functionality of the `httpd` daemon. The modules contain source code that provides the many different functions of Apache. Users can explicitly choose which modules to use at compile time by specifying them in the configure stage. Inserting modules one at a time at the command prompt allows specification of the modules to compile. Be careful; the more modules added, the higher the binary size, and there may be a slight degradation in performance as well. There is another way to specify modules in Apache. Inside the `src` direc-

tory of the Apache source tree, there is a file named `Configuration`. Toward the bottom of the file are module definition lines that reflect the default configuration. Specify which modules to use by uncommenting or by deleting the number (#) symbol on the lines. To add new lines or to modify existing lines, use the following format:

```
AddModule  /path/to/module/module_name
```

The following example is taken from an active Apache install and represents a logging facility that has been enabled by default:

```
AddModule modules/standard/mod_log_config.o
Adding Modules
```

Other options that are frequently used with `configure` are those that relate to modules. Apache comes with a number of modules that increase its functionality. Some of the modules are installed and enabled by default; others need to be selected at compile time. As mentioned earlier, there are three methods that can be used to enable modules. How a module is implemented is determined by how it is enabled with `configure`.

One module that comes with Apache but is not enabled by default is the `mod_info` module. This module allows access to detailed information about the server through a convenient Web interface. To enable it as a static module, invoke configure as follows:

```
$ ./configure --enable-module=info
```

Here are the options to enable the same module with DSO support:

```
$ ./configure  --enable-module=info \
               --enable-shared=info
```

If there is more than one option, using the line break escape character (\) will allow everything to fit on the command line in a coherent manner.

To simply experiment with the available options without having to specify every feature, include most modules as DSO modules with the following options:

```
 $ ./configure  --enable-module=most \
                --enable-shared=max
```

For details about the many available arguments for use with `configure`, read the `INSTALL` file in the main source directory.

Configuration Files

There are several configuration files within the Apache source tree. It is strongly recommended that these not be altered. Users may want to read some of them to see how the install works, but altering them could have unpredictable results on how things are installed. For reliable compiles and installs, customization should be done strictly with `configure` options. After invoking `configure` with or without options, there will be some text output as the scripts prepare the sources. Most systems that have been installed with a `developer` option should have everything needed to compile. Otherwise, the configure script will stop with an error that provides hints as to what might be wrong. Use the error message to determine what additional components should be installed.

CGI Scripts

Turning on `cgi` scripts for the entire server poses a security threat. Someone can create a `.cgi` file in a location other than the Web directory or another specified location; he or she can execute the script at a later date unbeknownst to the Webmaster. In short, it is better to allow `.cgi` files in only one directory. It is much easier to monitor one directory than an entire server.

The following subsections outline a few directives for enabling CGI scripts with which you should become familiar.

ScriptAlias Directive

The `ScriptAlias` directive defines a directory where CGI scripts are expected to be. Any script placed here, with proper permissions, will be available to any directory:

```
ScriptAlias /cgi-bin/ Ò/usr/local/share/apache/cgi-bin/"
```

AddHandler Directive

The `AddHandler` directive tells Apache the extension to look for on CGI scripts.

Processing of Requests

Requests from the network proceed through a series of stages inside Apache. At each stage, modules may deal with the request. Each hook can

succeed, defer to another module, or return an HTTP error response. In addition, they can modify the request object as it is seen by later stages.

In order, these stages are as follows:

1. URI to file name translation
2. Authentication
3. Authentication access checking
4. Nonauthentication access checking
5. Determining MIME type
6. Fixups
7. Sending the response
8. Logging the request

These stages match the Apache documentation phases.

Modules may start subrequests, in which Apache asks itself what would happen in response to a client request. For example, the module that produces directory indices (`mod_autoindex`) performs a subrequest to find information about each of the files in a directory.

Separate from subrequests, Apache can redirect the request to a different location. Apache does this either as an internal redirect, in which it generates the content that would have been produced by the new request, or by an external redirect, in which Apache sends a response code to the client identifying a different location to use.

Installation Summary

The steps for installing Apache are similar to those used for other applications and servers. Apache configuration will be discussed in greater detail later in this chapter. Following are the general steps for installing Apache:

- If it is in source form, verify the file.
- Extract the archive to a temporary directory.
- Configure it.
- Compile it.
- Verify the install on the local system and then the network.
- Add content.

After installing any new product, it is a good idea to stay informed about improvements to the product and about any issues that might arise. The Apache organization publishes a mailing list for just that purpose. Send a message (subject and content will be ignored) to announce-subscribe @apache.org. You will receive a confirmation message with instructions on how to validate your subscription. This mailing list is one-way, for announcements only, and messages will be received directly from Apache. One aspect of any extremely popular product is that many people are constantly working to find ways to attack it and a comparable number are working to defend against such attacks. The Apache organization has one of the best track records in the software industry for notifying users about problems and their solutions.

Configuring Apache Logs

Apache logs are in the Common Log Format (CLF) by default. The fields Apache tracks in the CLF are host, ident, authuser, date, request, status, and byte.

host

This is the IP address from which the request came.

Identity Check

Apache can query the client's identd (see RFC 1413) on the incoming http connection. To turn on this feature, edit the `httpd.conf` file and add:

```
IdentityCheck On
```

NOTE **identd checks can be extremely slow, so this feature should not be used on a busy Web server. identd checks can also be faked, so the information should not be completely trusted.**

authuser

If the requested document requires access authentication, the authenticated user id is placed in this field.

date

This field contains the date and time that the request was initiated. The proper format is: 05/May/2000:12:08:19 -0700.

request

This is the actual HTTP request that was received from the client, enclosed in quotes.

status

This is the three-digit HTTP status code returned to the client from Apache. Of these, the most common are 200, 202, 301, 302, 400, 403, and 404. Common status codes are shown in Table 1.2.

bytes

This is the number of bytes in the response sent back to the client, not including the response headers.

Alias

The `Alias` directive is a way of creating the equivalent of symlinks for our URLs.

Authentication—mod_access and mod_auth

The `Alias` command allows modification of the response to a request. A commonly used reference can be placed at an actual location, which is different from where the server seems to be getting it.

To format the alias command, use:

```
# Alias fakename realname
Alias /icons/ Ò/usr/local/share/apache/icons/"
Alias /gifs/ /web/ncsa/gifs
Alias /gif/ /web/ncsa/htdocs/gif/
```

Table 1.2 Status Codes and Their Meanings

CODE	MEANING
200	Everything is working great
202	Accepted
301	Moved permanently
302	Moved temporarily
400	Bad request
403	Forbidden
404	Not found

The mod_access and mod_auth modules provide the authentication functions of Apache. The way Apache authentication works is really quite simple. First, the client sends a name with a password to Apache. Then, Apache decides whether the host is allowed access to the resource. If so, Apache checks to see if the client has access by verifying the name and password. For instance, to prevent access to a /programs/ names.html file from anyone that is outside hello.com, simply type the following in httpd.conf:

```
<Location /programs/names.html>
Order deny, allow
  Deny from all
  Allow from .hello.com
</Location>
```

In recent versions of Apache, the file names are relative to the server root, or the file name is considered absolute. If the file name begins with /, it is said to be absolute.

Modules

The ability to add modules to Apache is key to its flexibility. Some modules are part of the basic functionality of the standard installation. About another 50 modules are included with the distribution that allow further customization. Some of these extra modules are enabled by default so that the server has more than the minimum functionality when first installed.

If one of the included modules does not meet a particular need, many authors have written additional modules that can be downloaded from the Internet. An organized listing of modules can be found at Apache's site:

www.modules.apache.org/

www.apache.org/dist/contrib/modules/

http://modules.apache.org/search

If using a module from the Internet, read the module's documentation very carefully! Some require Apache to be configured in a way that is specific and unique to that module.

Configuration Methods

When Apache is installed, configuration files are placed into the /configuration directory, whose location is defined at compile time.

Compile-time settings use the `-V` command-line flag. Use `httpd -V` to see compile-time settings of your installation:

```
Server version: Apache/1.3.14 (Unix)  (Red Hat/Linux)

Server built:   Mar  1 2000 13:37:34
Server's Module Magic Number: 19990320:7
Server compiled with....
 -D EAPI
 -D HAVE_MMAP
 -D HAVE_SHMGET
 -D USE_SHMGET_SCOREBOARD
 -D USE_MMAP_FILES
 -D USE_FCNTL_SERIALIZED_ACCEPT
 -D HTTPD_ROOT="/usr"
 -D SUEXEC_BIN="/usr/sbin/suexec"
 -D DEFAULT_PIDLOG="/var/run/httpd.pid"
 -D DEFAULT_SCOREBOARD="/var/run/httpd.scoreboard"
 -D DEFAULT_LOCKFILE="/var/run/httpd.lock"
 -D DEFAULT_XFERLOG="/var/log/httpd/access_log"
 -D DEFAULT_ERRORLOG="/var/log/httpd/error_log"
 -D TYPES_CONFIG_FILE="/etc/httpd/conf/mime.types
 -D SERVER_CONFIG_FILE="/etc/httpd/conf/httpd.conf"
 -D ACCESS_CONFIG_FILE="/etc/httpd/conf/access.conf"
 -D RESOURCE_CONFIG_FILE="/etc/httpd/conf/srm.conf"
```

Apache finds its settings in the configuration file `httpd.conf` (defined by the compile-time variable `SERVER_CONFIG_FILE`). Override the location of the configuration file with the `-f` command-line flag:

```
# httpd -f /etc/httpd/conf/httpd_alternate.conf
```

NOTE **These flags are particular to Apache 1. and will change with future releases of Apache. Please visit www.apache.org for updates.**

Other `httpd` runtime flags include the following:

```
# httpd -h
Usage: httpd [-D name] [-d directory] [-f file]

             [-C "directive"] [-c "directive"]

             [-v] [-V] [-h] [-l] [-L] [-S] [-t] [-T]
```

Starting httpd at Boot

Most users want Apache to start automatically when the system boots. Each Linux distribution has a different method of doing this; try to follow the conventions if possible. Some of the RPM and DEB packages may automatically add Apache to the startup scripts; others may not.

On a Red Hat 6.2 system, the RPM will probably not set Apache to start unless it was installed at the same time as the OS. To enable it at startup, use the ntsysv program. This program allows the user to determine which system services should be started at boot time.

If the proper way to start Apache at boot time cannot be determined, try adding it to the local startup script. Usually this will be found in `/etc/rc.d/rc.local`. Assuming Apache has been installed in `/www`, add the following entry:

```
/www/apachectl start
```

Setting Up Apache

The majority of the Apache setup is done via the `httpd.conf` file. If Apache is running during configuration changes, certain steps must be taken for the configurations to take place. These steps are outlined in the sections that follow.

Starting and Stopping Apache

Whenever the user alters the server, such as modifications in `httpd.conf`, Apache should be restarted for the changes to take effect. Apache comes with a script called `apachectl` that can be used for this purpose. `apachectl` works much like the service initialization scripts that most distributions have in the `/etc/ rc.d/init.d` directory. Specify `stop`, `start`, and `restart` to start or stop the Apache server.

Starting httpd Manually

To use `apachectl`, provide the path to the script. Adjust the search path to make `apachectl` more convenient to use. Use a command-line option to tell the script what to do. With no options, `apachectl` will display the help screen as shown:

```
usage: /usr/sbin/apachectl (start|stop|restart|fullstatus|status|graceful|
     configtest|help)


start          start httpd
stop           stop httpd
restart        restart httpd if running by sending a SIGHUP or start if not running
fullstatus     dump a full status screen; requires lynx and mod_status enabled
status         dump a short status screen; requires lynx and mod_status enabled
graceful       do a graceful restart by sending a SIGUSR1 or start if not running
configtest     do a configuration syntax test
help           Show this screen
```

fullstatus

Lynx and the `mod_status` module are required for `fullstatus` to provide a detailed description of the server status.

The result of the `apachectl fullstatus` command is as follows:

```
# apachectl fullstatus
   Apache Server Status for localhost.localdomain
   Server Version: Apache/1.3.14 (Unix) PHP/3.0
   Server Built: Dec 4 2000 09:46:50
     ____________________________________________________________
   Current Time: Thursday, 28-Dec-2000 19:34:56 PST
   Restart Time: Thursday, 28-Dec-2000 19:34:18 PST
   Parent Server Generation: 0
   Server uptime: 38 seconds
   Total accesses: 1 - Total Traffic: 2 kB
   CPU Usage: u.01 s0 cu0 cs0 - .0263% CPU load
   .0263 requests/sec - 53 B/second - 2048 B/request
   1 requests currently being processed, 4 idle servers
_W___.....................................................
.....................................................
.....................................................
.....................................................

   Scoreboard Key:
   "_" Waiting for Connection, "S" Starting up, "R" Reading Request,
   "W" Sending Reply, "K" Keepalive (read), "D" DNS Lookup,
   "L" Logging, "G" Gracefully finishing, "." Open slot with no current process

   Srv PID Acc M CPU SS Req Conn Child Slot Client VHost Request
   0-0 9886 0/1/1 _ 0.01 38 88 0.0 0.00 0.00 127.0.0.1
   localhost.localdomain GET /server-status HTTP/1.0
   1-0 9887 0/0/0 W 0.00 38 1191647253 0.0 0.00 0.00 127.0.0.1
   localhost.localdomain GET /server-status HTTP/1.0
     ____________________________________________________________
```

```
Srv Child Server number - generation
PID OS process ID
Acc Number of accesses this connection / this child / this slot
M Mode of operation
CPU CPU usage, number of seconds
SS Seconds since beginning of most recent request
Req Milliseconds required to process most recent request
Conn Kilobytes transferred this connection

Child Megabytes transferred this child
Slot Total megabytes transferred this slot
```

```
Apache/1.3.14 Server at localhost.localdomain Port 80
```

configtest

Use `configtest` to have the `httpd` configuration files parsed without restarting the server:

```
# apachectl configtest
Syntax [ OK ]
Checking configuration sanity for httpd: [ ok ]
```

graceful

Use `graceful` to gracefully restart Apache. When the user restarts with `graceful`, Apache will wait until it has served all pending requests before restarting, avoiding any annoying hard-stop interruptions users would experience with `apachectl stop`.

Default Index

When the `Indexes` option is enabled (`Options Indexes`), a directory with no default page displays as a directory listing. These commands allow considerable customizing of the directory's appearance, including choice of icons and a short description after each file.

The format to enable the indexing options is as shown here:

```
IndexOptions FancyIndexing
AddIconByEncoding (CMP,/icons/compressed.gif) \ x-compress x-gzip

AddIconByType (TXT,/icons/text.gif) text/*
AddIconByType (IMG,/icons/image2.gif) image/*
AddIconByType (SND,/icons/sound2.gif) audio/*
AddIconByType (VID,/icons/movie.gif) video/*
```

```
AddIcon /icons/binary.gif .bin .exe
AddIcon /icons/binhex.gif .hqx
AddIcon /icons/tar.gif .tar

AddIcon /icons/folder.gif ^^DIRECTORY^^
AddIcon /icons/blank.gif ^^BLANKICON^^

DefaultIcon /icons/unknown.gif
```

A directory listing can include more meaningful descriptions of file types, based on file extensions, as shown in the following example:

```
AddDescription "GZIP compressed document" .gz
AddDescription "tar archive" .tar
AddDescription "GZIP compressed tar archive" .tgz
```

Presence of a `README` file or `README.html` (and equivalent `HEADER` files) will cause Apache to display these files in a directory listing situation, allowing further customization of a plain directory listing.

The format used to display either a `README` file or `README.html` in a directory listing is as follows:

```
ReadmeName README
HeaderName HEADER
```

The following is the format for the directory listing to exclude certain file name patterns by inclusion in an `IndexIgnore` list (note that encoding types can be attached to files):

```
IndexIgnore .??* *~ *# HEADER* README* RCS CVS *,v *,t
AddEncoding x-compress Z
AddEncoding x-gzip gz
IndexOptions FancyIndexing
```

Basic Configuration

Some errors may have appeared when Apache was installed in Module 3. If errors did arise, they will be addressed in the following pages. If there were no errors, the user may notice the default index when visiting http://localhost.

The default index has a link to the official Apache documentation. We will be referring to the local copy throughout this chapter, but it can also be found on the apache.org Web site.

Different distributions perform different functions. The default install is similar to the following:

```
  :/etc/httpd/conf# ls
access.conf       httpd.conf.default  mime.types
srm.conf.default  access.conf.default  magic          mime.types.default httpd.conf
magic.default     srm.conf
```

These are the vanilla configuration files. In the past, Apache used a three-configuration file style where all general options were in `httpd.conf`. All resource-dependent configurations such as virtual servers, directories, document root, and all the directives and options that specify where files are in the file system as they relate to Apache were in `srm.conf`. Anything relating to access of these resources, such as blocking people from certain directories and turning on authentication, goes into `access.conf`. Three configuration files were confusing; all Apache configurations could go into any one of the files.

This outdated version of configuration filing was completely arbitrary. The system could have a zero length `httpd.conf` and `srm.conf` with all configurations in `access.conf`, and the system would perform the identical function. The main reason for this was for compatibility with the NCSA 1.3 Web server. Since the release of version 1.3.4, configuration filing has moved away from this; now the default uses only the `httpd.conf`. If the user needs a distributions' package, the configurations in the `httpd.conf` file should be checked. These lines, which are commented out by our default installation, are the ones to look for on the system to determine if Apache is using `srm.conf` and `access.conf`:

```
#ResourceConfig conf/srm.conf
#AccessConfig conf/access.conf
```

As of version 1.3, the files can simply be deleted; however, you may want to explicitly set them to zero by doing something similar to the following:

```
ResourceConfig /dev/null
AccessConfig /dev/null
```

For purposes of this course, it is important to understand that these files may or may not exist, and they may or may not be used. The user can start Apache with the `-f` option to specify the configuration file; therefore, the `rc` scripts should be checked as well to verify which `httpd.conf` file is being called. We will use only the `httpd.conf` in `/etc/httpd/conf`.

Usually, the default installation will work without any problems. If it does not, check the error log for a hint to the problem areas. To find the error log, search the `httpd.conf` file for `ErrorLog` (`grep -i error-log httpd.conf`). In our installation it is located in `/var/log/httpd`. We do not recommend leaving the server configured with the defaults, especially without understanding some of the consequences. Therefore, we will cover a few aspects to monitor.

Basic configuration of Apache is accomplished by modifying the directives in the `httpd.conf` file.

For a list of these directives, see www.apache.org/docs/mod/directives.html or start the server and follow the documentation link from http://localhost to the `Run-Time` configuration directives, found at http://localhost/../manual/mod/directives.html. Only a few of them will be mentioned here. Start by editing the `httpd.conf` file. Some of the first variations to explore are listed next with the default entries from our source installation.

Port 80

We noticed in our tests that sometimes the Port directive was incorrect or set to port 8080. This setting could cause problems, so double-check this first. If problems still occur, set the `Listen` directive to 127.0.0.1:80; however, this should not be necessary.

ServerType stand-alone

The `stand-alone` option is the default method and typically the only one that is used anymore. In `stand-alone` mode, servers are ready and waiting for connections. Not only will the user probably want to run the server this way, but it is also recommended for busy sites. From here, the number of servers that Apache starts and keeps running can be configured. This number will allow fine-tuning of configurations to meet specific requirements.

The inetd option is another ServerType option, but it is not commonly used anymore. The Apache documentation notes that the inetd option does not always work properly, but it does remain as an option. Using inetd will cause a decrease in performance because the httpd server will launch when the user receives a request on port 80 (or whatever port is set in inetd.conf) and then shut down after it serves the connection.

StartServers 5

This directive sets the number of servers that start when the httpd server launches. The default is 5, but for busier sites, increase StartServers to approximately 40. In the default case, httpd starts five server processes immediately ready to serve requests. The following directives will help fine-tune the way Apache manages these server processes.

MinSpareServers 5 and MaxSpareServers 10

These two directives allow the user to dynamically regulate the amount of server processes that are running at any given time. The directives are relatively straightforward. With the default configuration shown previously, we can be sure that there will always be at least 5 server processes available. When those are exhausted, the root httpd process will start more servers to maintain 10 idle server processes. These 2 directives work together to keep the operating ranges adequate for the server environment; however, this does not go totally unregulated.

MaxClients 150

The MaxClients directive is used to prevent the server from becoming overloaded. If there are more requests than a server can handle, it can eventually crash. This directive lets the user set a limit to the number of clients that can simultaneously connect. Once MaxClients is reached, Apache locks out additional requests.

MaxRequestPerChild 0

This directive appears to default to 0 despite other documentation. The user may want to change this to something other than 0, which means unlimited. This directive limits the number of requests that a server process handles before it dies. If there are any unknown memory leaks,

allowing a process to handle unlimited requests can obviously cause problems. If a module has no memory leaks, then `MaxRequestPerChild` can remain at 0. If the user is using experimental modules that do have memory leaks, a limit could be beneficial. Something between 100 and 500 (depending on how busy the server is) should be sufficient.

With the default configuration, the number of server processes can fluctuate between 5 (`MinSpareServers`) and 150 (`MaxClients`), while using the `MaxSpareServers` to determine how many of the idle processes should be killed.

How does the user determine how to set these directives? If `mod_status` is installed or compiled in, do the following to help determine operating ranges. The `mod_status` module should be loaded if the user followed the `./configure` options we used. To determine this, use the `-l` option with the `httpd` binary. Here is a sample of what may be included with a default installation:

```
COMPILED-IN MODULES
http_core.c
mod_env.c
mod_log_config.c
mod_mime.c
mod_negotiation.c
mod_status.c
mod_include.c
mod_autoindex.c
mod_dir.c
mod_cgi.c
mod_asis.c
mod_imap.c
mod_actions.c
mod_userdir.c
mod_alias.c
mod_access.c
mod_auth.c
mod_so.c
mod_setenvif.c
suexec: disabled; invalid wrapper /usr/sbin/suexec
```

ServerAdmin root@yourhostname.yourdomain.com

This configuration should have been set at compile time to root. Most Web sites change this to something similar to webmaster@yourdomain.com to

help identify the source of the e-mail. Modify the ServerAdmin to meet your needs.

ServerName yourhostname.yourdomain.com

This configuration is commented out by default, but the user may want to specify ServerName to obtain more reliable results with redirection. Use a valid DNS name for your domain.

UserDir public_html

Set the user directories here. This is relative to a system's $HOME environment variable. To set the absolute path, use the preceding /. There is a great deal that can be done with the UserDir directive. Please consult the online documentation for additional configuration options. Explicitly disable UserDir for the root user: UserDir disabled root.

Document Directory Configuration

```
DocumentRoot /home/httpd/html
```

This code is the default document root from our source installation based on the Red Hat layout. `DocumentRoot` is basically the root directory of a Web site or Web-space and is designated as such that its includes are used for a number of virtual domains. Every directory is controlled by a directory block that applies to its parent directory.

Therefore, the default for the root is strict:

```
<Directory />
    Options FollowSymLinks
    AllowOverride None
</Directory>
```

Options Directive

There are several options that can be set to determine the server features that are available on a per-directory basis. They are `ExecCGI`, `FollowSymlinks`, `Includes`, `IncludesNOEXEC`, `Indexes`, `MultiViews`, and `SymLinksIfOwnerMatch`. If the user does not set any options, the default is `All`.

NOTE **`MultiViews` is not included as part of All.**

Using the + and – will allow modification of options previously set on parent directories. A few are discussed here:

`All`	Everything except `MultiViews` is set if the user employs the All.
`ExecCGI`	ExecCGI allows the execution of CGI scripts.
`FollowSymlinks`	This allows the server to follow symlinks. Be careful when this is on. For example, if the user has a symlink to `/` or `/etc` with `FollowSymlinks` set, Apache will serve these directories to the clients.

Once Apache has been installed, it must be properly operated and maintained. The responsibilities of running an Apache server will be discussed in the next chapter.

CHAPTER

2

System Administration

Objectives

- State the common responsibility of a Webmaster.
- List basic steps to Web server security.
- Describe the functionality of MySQL.
- State the advantages of keeping logs.
- Compare/contrast the use of Addhandler and ScriptAlias.
- Compare/contrast static and dynamic modules.
- Describe ab's benchmarking method.
- List the reasons to use a redirect script.
- Describe the methods used to run multiple sites from one machine.
- List the steps that `httpd` takes when accepting a connection.
- List the reasons why aliases are useful.

Theory of Operation

Now that Apache is installed, the administration of the Apache server and modules, as well as the hardware and operating system that is running the Apache server, must be continued. This section will discuss the various roles and responsibilities a system administrator or Webmaster will have while configuring and running the Apache server.

Being a Webmaster

The title of Webmaster carries many different duties, all of which will be the responsibility of one person at some of the smaller Web installations. In a larger installation, different parts of the task are likely to be divided among several different people. The main groups of duties are these:

System administrator. Having total authority over the system.

Configuration management. Configuring the Web server.

Content management. Controlling design and deployment of the Web pages.

The system administrator has responsibility for the integrity of the system itself. This person is suspicious of changes to the system and is ever vigilant against intrusions from outside or from ordinary users exceeding their boundaries. The system administrator is in charge of the base operating system and usually the hardware as well.

Within the Web server's space, there are settings specific to each Web server, the domain of the Webmaster. This Webmaster is interested in protecting the server more than the system, and yet changes made to the server configuration files can cause problems for the system administrator. This manager must have awareness of the consequences at this level specific to Web operation and beyond the knowledge of a general Unix system administrator. Though the Web server daemon must be started by someone with root privilege, this person has no other needs for root privilege. Integrity of the Web server configuration files is the highest priority of this task.

There is yet another Webmaster—one who trades in content and process more than configuration. This person works with the artists who create the Web pages and directs the structure of the Web site. This person depends on the configuration being correct and makes requests based on the needs of the site. The tree of the Web site is built by this person. Direct contact

with building the Web server and setting the configurations is not central to this job, though this person will need to request changes when new features or new revisions become available.

In an individual's site or a moderately sized corporate site, these three conflicting personalities can be embodied in a single individual, or all duties can be shared by several people. In a larger site, they can be handled by two, three, or more persons, with the root user possibly being a different person than the Webmaster. The lines that separate the responsibilities can vary from one installation to another. At an Internet Service Provider (ISP), the provider could perform the first job or the first and second jobs (the third job will fall mainly with the end client). With modern virtual system methods, an ISP could give control of an entire "virtual system" to a client or just the configuration and content files or content files alone. If you are in charge of a system that is shared by many users, you will have to decide where the responsibilities lie and what powers to give to each level of Webmaster.

When the system is controlled by a single organization, the lines of division are much less critical. At a corporate site, the system administrator might give up root privileges to the Web managers, either temporarily or for a specific system dedicated to Web use.

When a system is shared among unrelated users, decisions at this level have more impact on system security. You will have to decide how much freedom to give to the Webmasters at your site and how much personal freedom to give up in order to maintain control. If you keep control over server configuration, you will have to be available to perform configuration changes. If you give up that control, you will want to do it in a way that doesn't give up the security of your other clients.

Specific commands can be opened up to the Webmaster or to the end user with tools like the following:

`sudo` (superuser-do or pseudo-su). Allows a user to run selected commands as superuser. Visit www.courtesan.com/sudo.

`chroot` (change root). Restricts access to another user's directory tree by changing the root directory that the user sees as the system root to a portion of the actual system root (probably just the user's personal directory). Visit http://hoohoo.ncsa.uiuc.edu/docs/tutorials/chroot.html.

`suexec`. Switches the effective user of a CGI script to that of the user account. Visit www.apache.org/docs/suexec.html.

When a group of Web sites is shared by the same organization, or perhaps different divisions or departments of the same corporation, the group of sites will probably be managed by the same person or group of people. There is no concern about information being shared between them or one of the sites being sabotaged by a partisan of another site. In another installation, perhaps in a public Web site hosting service, the adjacent Web site may be occupied by a total stranger. In this latter instance, there is a need for separation between users of each site. Control of the Web server configuration file becomes a sensitive issue.

Of course, there are several solutions. A separate Web server daemon for each site could allow each Web client to manage its own configuration file and not have access to its neighbor's. Restricting management of the configuration file to the site administrator is another alternative.

Tuning

Running Apache on Linux works quite well, even under high server loads. There are a number of settings and directives that can be adjusted to further enhance performance. Apache 1.3 introduced a number of performance-enhancing changes to the code and default configuration. As a result, not much needs to be modified, compared to previous releases. A few things still remain that can enhance performance further, including optimizing Linux for increased speed and process capacity.

Before optimizing, think about what the server needs to do. Will other services, such as mail, FTP, and databases, be running on the same box? Will enough images be served to require a separate server?

Many large Apache installations separate the data server from the HTTP server. If planning a database-driven Web site, it pays to look at that option. Freeing the Web server from working with a database helps it focus on delivering static html content.

Preparing Apache

The Apache developers have increased the performance of the server with the release of 1.3.14. As a result, performance is nearly optimal. Some additional adjustments that can increase performance have not been included by default because their optimal settings can be determined only by the administrator at a given location.

Allow and Deny

The `Allow` and `Deny` directives can slow things down, depending on how they are used. If the target location is an IP address, there is not a problem. If the target is a domain name, it will take some time to resolve the domain name into an IP address. Apache incorporates spoofing protection with these directives, so a reverse look-up also occurs. Use IP addresses with these directives to avoid the look-up penalty. Deny access starting at the system root directory, and allow access selectively where needed. This way access permissions will not have to be checked with each request.

MaxClients

This directive controls how many visitors Apache will handle at once. If the number of requests exceeds this value, clients will be locked out. This is a safety feature to keep Apache from taking the entire machine along as it absorbs all available system resources. Check to see if swapping is taking place when `MaxClients` is reached, and adjust the value accordingly. If there is no swapping at this point, the value can be increased. Decreasing it can reduce swapping. Adding RAM can also reduce swapping and allow increasing `MaxClients`, resulting in more simultaneous connections.

The upper limit of 256 is hard-coded at compile time. Increasing beyond the current limit will require a recompile, after putting the desired value in the `HARD_SERVER_LIMIT` variable that is defined in `httpd.h`.

TCP/IP Version

Be sure to have the latest version of the system kernel. The serving of HTTP invalidates many assumptions built into Unix kernels from as recently as 1994 or 1995.

Host Name Look-Ups

Before Apache 1.3, host name look-ups defaulted to `On`. This required a completed DNS look-up before Apache could finish a request. With Apache 1.3, the feature is defaulted to `Off`. Now, when the allow from domain or deny from domain features are used, a double reverse DNS look-up will be performed (a reverse followed by a forward to ensure that the reverse is not being spoofed). If an IP address is used instead of a domain name, this penalty will not occur.

NOTE **These directives can be scoped so that look-ups are performed only on requests that match specific criteria. The following example code disables look-ups, except for `.html` and `.cgi` files:**

```
HostnameLookups off
<Files ~ "\.(html|cgi)$">
    HostnameLookups on
</Files>
```

There is another alternative available within CGIs. Consider using the `gethostbyname` call within the specific CGI that requires a DNS look-up.

FollowSymLinks and SymLinksIfOwnerMatch

Within the Web directory tree, any place that does not have `Options FollowSymLinks` or does have `SymLinksIfOwnerMatch`, Apache will have to issue extra system calls (one extra call per file name component). For example:

```
DocumentRoot /etc/apache/htdocs
<Directory />
    Options SymLinksIfOwnerMatch
</Directory>
```

If a request is made for `/index.html`, Apache will have to perform an `lstat(2)` on `/etc`, `/etc/ apache`, `/etc/apache/htdocs`, and `/etc/apache/htdocs/index.html`. These requests are never cached, so they will occur on every request. If not using symlinks security checking, the following configuration will provide a faster performance:

```
DocumentRoot /etc/apache/htdocs
<Directory />
    Options FollowSymLinks
</Directory>
<Directory /etc/apache/htdocs>
    Options -FollowSymLinks +SymLinksIfOwnerMatch
</Directory>
```

Any `Alias` or `RewriteRule` sections outside the Web directory tree will require similar treatment. For best performance (but no symlink protection), set `FollowSymLinks` everywhere, and do not use `SymLinksIfOwnerMatch`.

AllowOverride

Wherever override is allowed, Apache will attempt to open an `htaccess` file. For example:

```
DocumentRoot /etc/apache/htdocs
<Directory />
    AllowOverride All
</Directory>
```

Every time an attempt is made to open `/index.html`, Apache will try to find and open `/.htaccess`, `/ etc/.htaccess`, `/etc/apache/.htaccess`, and `/etc/apache/htdocs/.htaccess`, whether they exist or not.

NOTE **These directives can be scoped so that look-ups are performed only on requests that match specific criteria. The following example code disables look-ups, except for `.html` and `.cgi` files:**

```
HostnameLookups off
<Files ~ "\.(html|cgi)$">
    HostnameLookups on
</Files>
```

Another alternative is available within CGIs. Consider using the `gethostbyname` call within the specific CGI that requires a DNS look-up, as in the following example:

```
FollowSymLinks and SymLinksIfOwnerMatch
```

Content Negotiation

One opportunity for a measurable speed increase is in the `DirectoryIndex` directive. Instead of the convenient wild card, give a complete list of options with the most commonly found choice first (or with the preferred one first, if several are present). Instead of:

```
DirectoryIndex index
```

use:

```
DirectoryIndex index.shtml index.html index.htm index.php
```

Within the Web directory tree, for any place that does not have `Options FollowSymLinks` or does have `SymLinksIfOwnerMatch`, Apache will have to issue extra system calls (one extra call per file name component). For example:

```
DocumentRoot /etc/apache/htdocs
<Directory />
    Options SymLinksIfOwnerMatch
</Directory>
```

Process Creation

Before Apache 1.3 was released, the `MinSpareServers`, `MaxSpareServers`, and `StartServers` directives had drastic effects on benchmark performance. This results from the difference between the way benchmarks are run and the way real-life requests are issued. In a benchmark run, an attempt at 100 simultaneous requests would begin the run. Before Apache 1.3, if there were more requests than servers, an additional server would be spawned each second until enough were present (this was to avoid swamping the server with spawning new processes). Starting with the default `StartServers` of 5, it would take approximately 95 seconds before enough servers were present to handle the test load. In a real situation, the ramp-up is gradual, and this technique is good enough; in a benchmark that might run 10 minutes, results can be misleading.

In Apache 1.3, the daemon will spawn one child process, wait a second, spawn two children, wait another second, spawn four, and so on, until it is spawning 32 new child processes per second. New child processes continue to be spawned at this high rate until `MinSpareServers` is satisfied or `MaxClients` is reached. Whenever more than four child processes are spawned per second, an entry will be made in the error log. If many such entries occur, consider tuning these settings. Use `mod_status` as a guide.

Process Death

By default, the `MaxRequestsPerChild` setting is 0, meaning that a child process will continue running as long as there is a load to justify it. Consider increasing very low numbers, such as 50. If running SunOS or an old version of Solaris, limit it to 10,000 because of memory leaks.

KeepAlive

When `KeepAlive` is in effect, children will spend time waiting for requests on an open connection. The default value of 15 seconds attempts to minimize the effect. There is a trade-off here between network bandwidth and server resources.

NOTE **Do not ever increase the `KeepAlive` value above 60 seconds, or most benefits will be lost.**

For more detailed discussions of performance issues, visit the following Web sites:

www.apache.org/docs/misc/perf-tuning.html
http://linuxperf.nl.linux.org/webserving/
www.kegel.com/c10k.html
www.kegel.com/mindcraft_redux.html
www.zabbo.net//phhttpd/

Introduction to Virtual Hosting

Name-based hosts require that the `NameVirtualHost` directive be placed above the host sections, and the directive must contain a valid network address.

IP-based or non-name-based hosts in `httpd.conf` need to be placed between an opening and closing `VirtualHost` tag pair, like the following:

```
<VirtualHost 192.168.1.3 >
...
</VirtualHost>
```

Each section encloses configuration information for a virtual host. Any configuration information unique to that host will be enclosed within this section. A name, instead of an IP address, could be used here but is not recommended for a number of reasons, including performance, reliability, and possible security risks.

Directories

Create a directory for each host that contains everything that is specific to that host. For single daemon configurations, a few directories contain the Web content itself and space for logs; add others as needed. Multiple daemon configurations require a complete Apache install for each host. This install means a complete Apache directory tree and configuration files.

Network Addresses

For some of the methods, you will need to create and configure additional IP addresses. This can be done with IP aliasing. For IP-based hosting, obtain valid Internet addresses. Name-based hosts share a single network address. Name-based hosts need to have the names registered.

Introduction to Apache Modules

As previously stated, Apache administrators have the option of installing module capability in Apache and deciding which modules to install. There are a wide variety of modules, some of which will be covered in this section. These include CGI, PHP, MySQL, and Perl. It is important to realize the capabilities and dangers of each module in order to be an effective Apache administrator.

What Is CGI?

According to the World Wide Web Consortium, the Common Gateway Interface (CGI) is a "standard for external gateway programs to interface with information servers such as HTTP servers." In practice, this means CGI is a method of allowing the execution of programs on Web servers and then passing data back to the Web client to be viewed.

When a Web browser requests a page from a Web server, the server must know whether that page is a CGI page. If it is, the server must execute it and then pass the returned content back to the Web browser. Although commonly called CGI scripts, these programs can be any type of valid program recognized by the Web server. This recognition process is shown in Figure 2.1.

When to Use CGI

Typical Web pages are static, meaning the content shown when a page is viewed does not change from time to time or based on environment. CGI

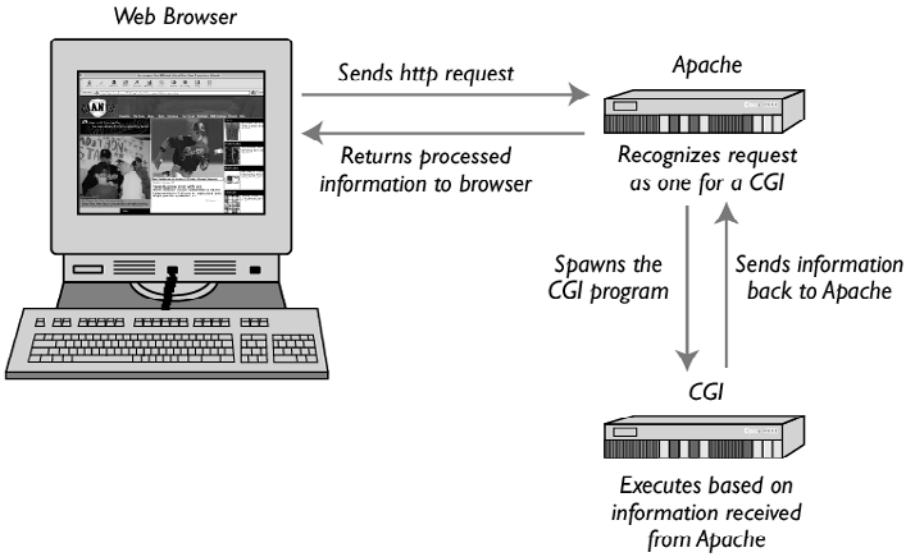


Figure 2.1 The Web browser, Apache, and CGI.

allows for interfacing (gatewaying) between the Web server and some other entity, such as a database, and allows the content returned to the browser to be dynamic based on certain conditions.

CGI was one of the first methods implemented to achieve dynamic Web pages and, because of its flexibility, is in wide use today. Other methods of creating dynamic pages include Java, JavaScript, VBScript, SSI, and inline scripting languages, such as PHP and ASP.

The flexibility of CGI comes from its ability to execute any program that the system itself can execute. A CGI script can be a shell or Perl script, a compiled C program, a python or tcl script, or almost any other language. It can also be tested outside of the Web environment because it is a program native to the operating system. For example, CGI programs can be executed directly from a shell.

The major drawback of CGI is also based on its nature as a program. Because it is a program, it must execute as one, which incurs the (sometimes severe) overhead of loading a new process into memory and executing it. The slight pause that occurs between executing a program on the command line and seeing its output, multiplied by thousands of hits per minute, can add up to significant slowdown in the responsiveness of a Web

site. While the script is processing, the impatient user browsing the site is staring at a blank browser window.

PHP—mod_php4

Although the PHP language can be used to write CGI scripts, it is more commonly installed as a module, substantially improving performance. The PHP language is tightly integrated with HTML. This integration allows insertion of common programming constructs into HTML code. It also enables sophisticated programs to be written in an HTML environment. One of the more interesting features of PHP is the ability to integrate closely with common database systems and embed SQL code within PHP.

PHP Mailing Lists and More

For more information about the PHP Development Team and PHP project please see www.php.net.

MySQL

MySQL is a database management system, which contains files that Apache can access through modules. This allows for Apache-driven Web sites to provide database-driven, dynamic, and user-specific Web pages.

Database Concepts

One of a computer's most powerful tools is the ability to store and retrieve data. Many techniques for retrieving data have been tried over the years, from linear search of plain-text files to Indexed Sequential Access Method (ISAM) tables. This search for effective retrieval methods has led to the development of database management systems (DBMS). Different classes of DBMS have been developed, including hierarchical, network, relational, and object. The relational model, formally described by E. F. Codd in the late 1960s (published in 1970), has been extremely successful and is the type of database that will be examined here.

A database consists of a set of data organized for easy retrieval and special software that is designed to retrieve it. In a relational database, the underlying concept is that everything is stored in tables. Although there are special technical terms for the components of a database, the more familiar common terms will be used in this discussion, with some reference to classical programming terms. A table consists of rows (records), which are divided into columns (fields). Each row in a table can be accessed and manipulated separately from all other rows, possibly using one or more key columns or even using keys built up from multiple columns. In classi-

cal programming, files are made up of records; in database programming, tables are made up of rows.

Besides this basic organization, which could describe common file formats used in programming, there is a set of rules that distinguishes the structure of a database system from a classical programming application. Among the rules is the concept that all characteristics of the data structure, as well as any changes to the data structure or content, are controlled through the DBMS and are reflected in tables. Values entered into a field can be constrained by rules enforced by the DBMS, not by the application. Any applications that access the data are kept separate from the data itself and go through the DBMS to reach the data. As a result, data integrity is in the domain of the DBMS, not the domain of the application. Any data item is referenced uniquely by its table, key, and column values. In this way, any redesign of the data structure will have no effect on the application. By defining a relationship between fields in different tables, it is possible to link tables together. Another important feature of a DBMS is that a single command can be issued through the DBMS interface that will affect all rows of a table, without the need for record-by-record processing. The abilities to relate data and to isolate the internal workings of the database from the application are two of the things that give a relational database its power.

Normalization of Tables

Normalization of tables in a relational database is the process of organizing the database into tables in such a way that the results of using the database are always unambiguous and predictable. Normalization consists of reducing the duplication of data items within the database according to a set of rules, often with the creation of additional tables. Normalization will ensure that a particular item is stored in a single location. If you update an item, any references to that item will show the update by logical design, not by any special action.

A database can be described in terms of its level of normalization, according to how many of the five rules of normalization the database's design follows (Boyce-Codd Normal Form, or BCNF, is a variation of third normal form, or 3NF). Typically, the optimum balance of speed and complexity occurs at 3NF or BCNF. Normalization beyond this level results in an increase in the number of tables, with an associated increase in complexity and a possible decrease in speed. Normalization is typically a refinement process after the initial exercise of identifying the data objects that should be in the database, identifying their relationships, and defining the tables required and the columns within each table.

SQL Databases

SQL (often pronounced sequel) stands for Structured Query Language. It is the best-known database language. It has been widely implemented and is available in almost every commercial relational database management (RDBM) system and many open source systems as well. SQL provides the abilities to create and maintain the database and to query the database. SQL is recognized by the American National Standards Institute (ANSI) as the standard language for relational database management systems. SQL is a comprehensive database language, with statements for data definition, query, and update. Although most database systems use SQL, many of them also have their own proprietary extensions that are available only on their system. A good RDBMS can perform all standard operations without having to call on the proprietary extensions.

Selecting a Database

Several database systems exist for Linux. While some are commercial packages, others are free (open source). Considerations when choosing a database include the amount of data to be stored, the type and frequency of transactions, security features, multi-user support, access method, speed, and accuracy. MySQL will be used for this lesson because of its ease of use and also because of the extensive selection of third-party tools available freely on the Web. As of June 2000, MySQL was released under the GNU GPL-license, which makes it free for use in commercial systems.

The main feature of MySQL is that it is multithreaded, optimized for speed and for use with large tables. MySQL gains its speed advantage at the cost of such desirable SQL features as triggers, subselect, and transaction processing. Developers, though, will gradually roll certain desirable features into the product over time. The intended use of the product is as a quick, fast implementation of simple applications.

For more information on MySQL, visit the Web site at www.mysql.com.

Several other database systems are available for Linux. Commercial packages include Oracle 8i, IBM DB2, and Informix SE.

The mSQL package is available as freeware at prices comparable to those for MySQL at www.hughes.com.au/ products/msql/. It is free for non-commercial use.

PostgreSQL is derived from the Postgres research database. It is a full-featured object-relational database system. It supports declarative queries in SQL, query optimization, concurrency control, and transactions. It is completely Open Source and released under GPL. PostgreSQL comes standard with many distributions and is quite powerful. It can be found at www.postgresql.org.

Database Design Considerations

Before beginning to implement a database system, users must analyze their needs by examining how many records of what size and kind will be generated over the life of the project and what kinds of operations are expected to be performed on the data. Also, they examine how the data will be accessed and how fast the connection will be. Transaction processing is a special task and places extra demands on a database system. After determining what system needs will be, users should examine different database packages and see which ones match the application.

After settling on a specific platform, design the database using whatever tools are available on that system to assist in the design work. Remember that part of the design is a security profile, a plan for which users will have access to which portions of the database.

After these steps, the database should be ready for data entry.

MySQL Architecture

MySQL uses a client/server architecture. In very simplified terms, a client/server architecture means that the data and all of the software necessary to manage the data (the server) is kept separate from the software that is requesting and using the data (the client). Also, a typical server is able to handle multiple clients. The MySQL distribution package includes both server and client components.

Because the client and server are not required to be on the same physical machine, it is common to have a variety of client systems accessing the same server and sharing the same data. There is no limit on where the systems can be located; either client or server could be anywhere in the world. This fact does not imply that anyone in the world can access data on a server without consent. The MySQL administration section covers how to secure a database, allowing access to the server from specific hosts and assigning specific access levels to different users. Finally, the multi-user nature of the client/server system requires concurrency control so that two users are not able to modify the same record at the same time.

MySQL System Administration

Once the database server is running and the database is in use, paying periodic attention to a few critical details can ensure that the data remains available to those who need it and closed to those who do not. Scheduling frequent backups and keeping transaction logs can help users recover quickly from disasters, either man-made or natural. A properly configured security profile can help control who can see the data and what they can do with it.

Starting and Shutting Down the Server

The `safe_mysqld` script will start the server safely with standard settings and can restart it after a system restart or a crash. To start a server with individual options, a user can employ the `mysqld` program directly, edit options into the `safe_mysqld` script, or use the `safe_mysqld` script as a model for building a customized startup script. The `mysqld` program is usually stored under the `libexec` directory of the MySQL base install directory. The general usage is:

```
Usage: mysql [OPTIONS] [database]
```

To start `mysql`, type:

```
[root@sair.linuxcare.com Apache]# mysql
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 7 to server version: 3.23.23-beta-log
Type 'help;' or '\h' for help.
mysql>
```

By typing `\h` at the `mysql>` prompt, the following general commands are displayed:

```
MySQL commands:
Note that all text commands must be first on line and end with the ';'.
Help           (\h)      Display this text
?              (\?)      Synonym for "help"
clear          (\c)      Clear command
connect        (\r)      Reconnect to the server. Optional arguments are db and host
edit           (\e)      Edit command with $EDITOR
exit           (\q)      Exit mysql. Same as quit
go             (\g)      Send command to mysql server
ego            (\G)      Send command to mysql server; display result vertically
print          (\p)      Print current command
quit           (\q)      Quit mysql
rehash         (\#)      Rebuild completion hash
source         (\.)      Execute a SQL script file. Takes a file name as an argument
status         (\s)      Get status information from the server
```

```
use           (\u)      Use another database. Takes database name as argument
Connection id:  7       Can be used with mysqladmin kill
mysql>
```

Type `mysql -h` to view available options for the `mysql` command. The MySQL command syntax is as follows:

```
# mysqld <options>
```

where options can be any of the previously listed values.

To shut down the MySQL server, use the `mysqladmin` command with the `shutdown` parameter:

```
# mysqladmin shutdown -p
```

Making Backups of Your Data

Backing up data is one of the first principles of survival when using a computer. A backup (if done properly) can protect data from a range of problems, from a system crash (is the power cord where someone might trip over it?) to an act of vandalism (is there a government building anywhere nearby that might be subject to a terrorist attack?) to a weather problem (how much snow did you say the roof can support?). All of these events have occurred in recent years; even the most reliable system might not be able to continue operating under extreme conditions. Backing up data and keeping the backup at a different location can make a world of difference in how long a system is down if something unexpected happens.

There are two ways to make a backup. Because MySQL uses common data files in the Linux file system (unlike some optional configurations on large commercial databases), a backup can be performed by copying all table files (`*.frm`, `*.MYD`, and `*.MYI`) at a time when the server is not updating anything. Updates can be prevented during the file system-level backup by doing a `LOCK TABLES/UNLOCK TABLES` set of commands. A `read` lock will allow queries to continue during the backup, while disallowing updates.

Backing up with the `mysqldump` command will allow updates to take place during the backup. To be able to capture the updates, first stop `mysqld` and then start it with the `--log-update` option. This example assumes that `log-update` is not already turned on:

```
# mysqladmin shutdown -p
# mysqld <Other Options> --log-update
# mysqldump --tab=/path/to/the/dir --opt --all-databases
# mysqladmin shutdown -
# mysqld <Other Options>
```

The `mysqldump` command will create a set of files in the directory pointed to by the `--tab=` switch for each table. `--opt` will cause the dump to create an optimized set of commands to rebuild the data. `--all-databases` backs up each database in the system to its own set of files. The `log-update` option will create log files with names like `hostname.n`, where `n` is a number that is incremented each time `mysqladmin refresh` or `mysqladmin flush-logs` is executed, the `FLUSH LOGS` statement is executed, or the server is restarted. These log files have the information that will allow replication of the changes to the database, which are made after the `mysqldump` process began. Be sure to study the documentation at www.mysql.org or one of its mirrors so that you are familiar with what goes on when any of these commands is issued.

If the database is lost, first try to recover with `myisamchk -r`. This should work in almost all cases. If `myisamchk` fails, try the following procedure:

Restore the original `mysqldump` backup.

Rerun the updates in the update logs by executing this command:

```
# ls -1 -t -r hostname.[0-9]* | xargs cat | mysql
```

The `ls/xargs` command will access the log files in the correct order, if there is more than one.

The example illustrates a full database backup. The `mysqldump` command will also allow a dump of a specific table from the database:

```
# mysqldump contact > backupfile
```

This line creates a file that will be able to rebuild the database. The file contains the SQL commands to create the tables and insert statements that insert the data into the tables. To add a drop table command to the SQL dump (SQL statements to drop the table before creating a fresh copy), specify the `drop-table` option:

```
# mysqldump --add-drop-table contact > backupfile
```

To get the specific options available on each copy of `mysqldump`, issue the following command:

```
# mysqldump --help
```

Table 2.1 shows some typical `mysqldump` command options.

To retrieve a database, read the backup file into the MySQL client program:

```
# mysql contact -p < backupfile
]
```

Table 2.1 mysqldump Command Options

OPTION	COMPLETE OPTION	DESCRIPTION
`-a`	`--all`	Includes all MySQL create options
`-#`	`--debug=file`	Outputs debug log; often this is d:t:o file name
`-?`	`--help`	Displays the help message and exits
`-c`	`--complete-insert`	Uses complete insert statements
`-C`	`--compress`	Uses compression in server/client protocol
`-e`	`--extended-insert`	Allows utilization of the new, much faster INSERT syntax
	`--add-drop-table`	Adds a drop table before each create
	`--add-locks`	Adds locks around insert statements
	`--allow-keywords`	Allows creation of column names that are keywords
	`--delayed`	Inserts rows with INSERT DELAYED
`-F`	`--flush-logs`	Flushes logs file in server before starting dump
`-f`	`--force`	Continues even if you get a SQL error
`-h`	`--host=hostname`	Connects to a specified host
`-l`	`--lock-table`	Locks all tables for read

This will create a new copy of the database and insert the data statements from the incoming file into the new database. In the preceding example, `contact` is the database name.

To back up (or transfer) specific records from a database, do a selective backup with a `select` statement:

```
mysql>SELECT * INTO OUTFILE 'file_name' FROM tbl_name
```

Restore the table with this:

```
mysql>LOAD DATA INFILE 'file_name' REPLACE ...
```

There is a risk of duplicate records unless a `PRIMARY KEY` or a `UNIQUE` key is set in the table. Use of the `REPLACE` keyword allows new records to replace old ones, when a new record has the same key value as an older record.

Managing User Accounts and Security Privileges

It is standard business practice that certain transactions must be available to some users and not to others. For example, the user who enters invoices may be a different person from the one who pays invoices. Another user can enter payroll data, while someone else can only look at it (although most are not permitted to look, either). The business rules of a task are reflected in the design of the corresponding database. Users of a database are granted privileges based on what type of operations they need to perform and what portions of the database they are permitted to access. Privileges can be granted to users either by directly manipulating the user table or by the preferred method of using the `grant` command.

Syntax for the `grant` command is:

```
GRANT privileges
```

on the user identified by `password` with `grant` option.

PRIVILEGES AND OPERATIONS ALLOWED	
`ALTER`	Alter tables and indexes
`CREATE`	Create databases and tables
`DELETE`	Delete existing records from tables
`DROP`	Drop databases and tables

INDEX	Create or drop indexes
INSERT	Insert new record into tables
SELECT	Retrieve existing records from table
UPDATE	Modify existing table records
FILE	Read and write files on the server
PROCESS	View information about threads running
RELOAD	Reload the grant tables or flush logs
SHUTDOWN	Shut down the server
ALL	All privileges
USAGE	No privileges

The more powerful privileges should be restricted to a short list of administrators.

Keeping Logs of All Transactions

Every transaction that occurs on the MySQL server retrieval can be logged at a later time. When starting the MySQL server daemon, specify the `--log` and `--log-update` parameters to tell the server to save a log. The `--log` option will turn on general logging; the `--log-update` option enables logging of changes on the database.

```
# mysqld --log --log-update
```

As transactions are sent to the server, the log files fill up, eventually to the point where they take all available space on that file system. With a busy server, this could happen quickly; with a less busy server, it will take more time. All the same, if the old transactions are not purged, the file system will eventually fill up.

One way to deal with the situation is to write a script that rotates the log files. Each time the MySQL server restarts, it creates a fresh log file. If a script (invoked by `cron`) is created that restarts the MySQL server, then deletes the oldest log file (or any log file over a certain age), the file system will fill up much more slowly.

Log files by using the `mysqladmin` command with the `flush-logs` parameter:

```
# mysqladmin flush-logs -p
```

This will clean out all of the log files. Be sure that any log files necessary for a restore have been backed up before running this command.

mod_perl

Perl is a high-level programming language written by Larry Wall. It is derived from the C language and contains lots of functions and tools for numerous tasks, such as system administration, database access, string manipulation, networking, and, most importantly, Web programming. The newest version of Perl is freely available at www.perl.com.

Perl has its own module system. Perl modules are similar to libraries in other programming languages. A module is a set of functions that provide additional functionality to the Perl language and can be included and used easily in Perl programs. Perl modules are stored in files with an extension of `.pm`, so `CGI.pm` would be a Perl module, while files without an extension, or those ending in `.pl`, are Perl programs. All Perl modules can be downloaded from the Comprehensive Perl Archive Network (CPAN), which contains all known Perl material and modules. CPAN can be found at the www.perl.com site and many mirror sites.

Apache's `mod_perl` module integrates a Perl interpreter with the Apache Web server. It contains Perl code that provides for an object-oriented interface to the new Perl-enabled Apache server. The `mod_perl` module contains lots of functions and classes that add great functionality to the Apache server. With `mod_perl`, it is possible to write Apache modules entirely in Perl. The major advantages of `mod_perl` are speed and the ability to deal with the inner workings of the Apache Web server. CGI scripts can also be run under `mod_perl`. Installation of `mod_perl` is fairly complex. Please do not try to do this without reading the `README` files that come with the module installation files. The simplest way to install is to use the `Makefile.pl` script with the right parameters.

`mod_perl` mailing lists and more are located at http://perl.apache.org.

Introduction to the Apache API

The Apache Application Programming Interface (API) is the method by which new pieces of code (i.e., Apache modules) integrate their handlers into the main flow of operation in Apache. Knowing the steps Apache follows when handling a request can help you to understand how Apache works.

This chapter will introduce the Apache API with a general overview. It will give information about how Apache handles Web requests (i.e., what is going on under the hood).

Modules

Modules are the first critical component of the Apache API. A module is a bit of code written to handle a specific task, such as logging, authentication, directory and alias handling, access control, content negotiation, and CGI and Server-Side Includes. Most of Apache is implemented as a module—the Apache core handles some pieces of the server but generally hands processing off to a module.

Each module can be compiled into Apache, similar to a statically linked library, or compiled as a dynamically shared object (DSO), similar to a shared library. Static modules become part of the `httpd` binary and are automatically recognized by the Apache core. DSOs each have a file on the file system and must be loaded by a `LoadModule` command in Apache's `httpd.conf` file in order to be initialized. The flow of operations is shown in Figure 2.2.

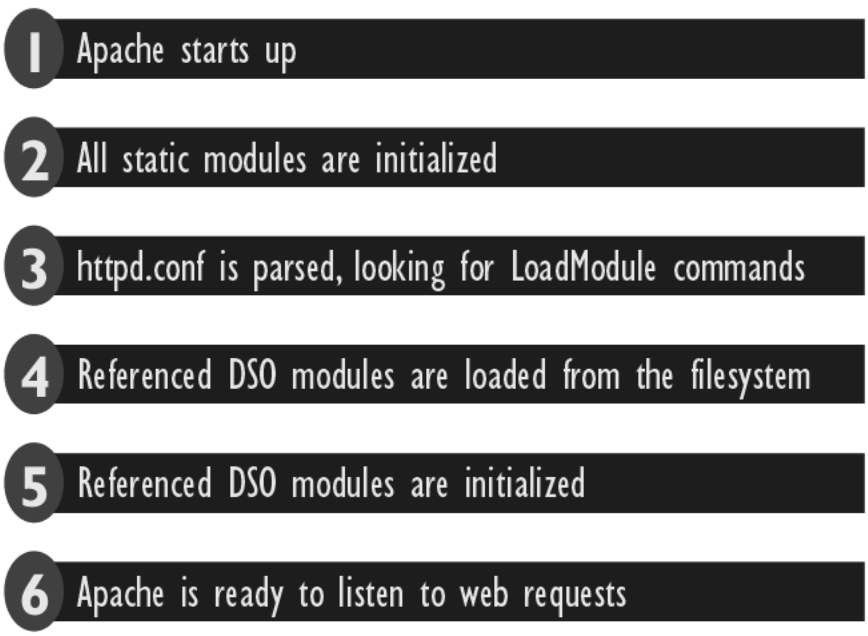


Figure 2.2 Apache flow of operations.

Phases

Phases are one of the three critical components of the Apache API. The Apache flow of operation goes through eight phases each time Apache receives a request to serve a Web page. A request flows from Phase 1 to Phase 8 in sequential order. When Apache receives a request for a document, it starts by passing it through Phase 1; by the time the request reaches Phase 8, the response has been sent and Apache waits for a new request. The eight phases of Apache are illustrated in Figure 2.3.

Handlers

A handler is a set of C functions that a module uses to perform a task. Inside the code of each module is a structure identifying what type of processes it can perform, as well as the code to actually perform it; this is a handler. Each module can have one or more handlers defined in it.

The handler is the part of the module that performs actions. If a module is written to process CGI scripts, the module will have a structure telling Apache which handler function to call when it sees a request for a CGI

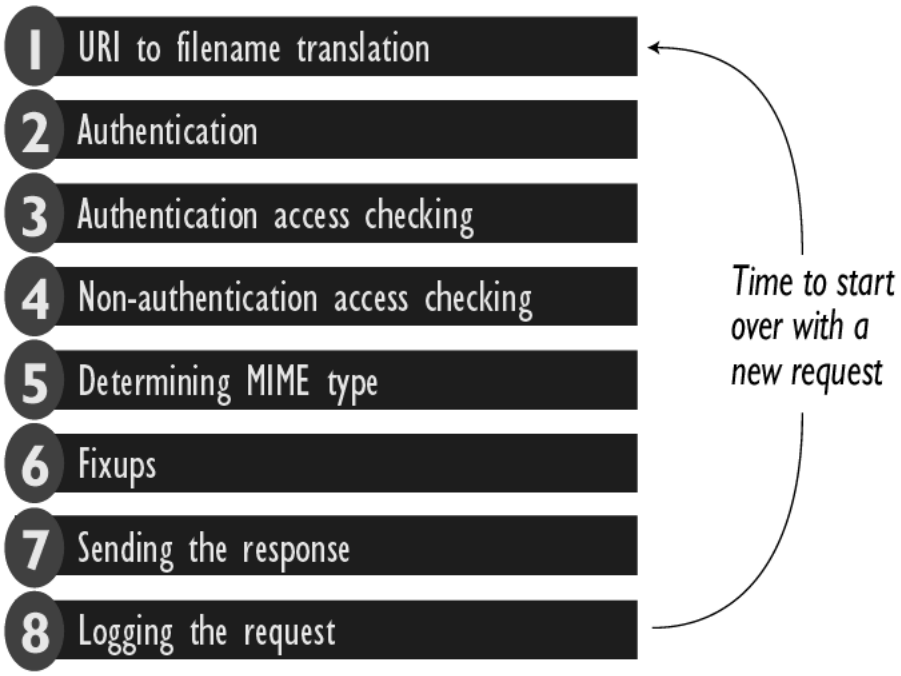


Figure 2.3 Apache phases.

script. Apache keeps a list of which handlers perform which functions and which modules are associated with each handler.

When each module is initialized, Apache looks at the module's handler structure(s) to determine what that module does. These structures tell Apache each phase for which they are appropriate, as well as how to call the handler when needed. Each handler has the option of declining the request or attempting to handle the request when called.

If the request is declined, Apache tries the next handler registered for that phase. If the request returns a success, the next phase is attempted. If it returns a failure, the request goes straight to the last phase: logging the request.

There are exceptions to this flow of operations. For instance, when a handler returns a success after handling a request, the request moves to the next phase, except for the Non-authentication Access Checking, Fixup and Logging the Request phases. In these phases, all handlers run as long as no failures are returned by previous handlers. Apache handlers are shown in Figure 2.4.

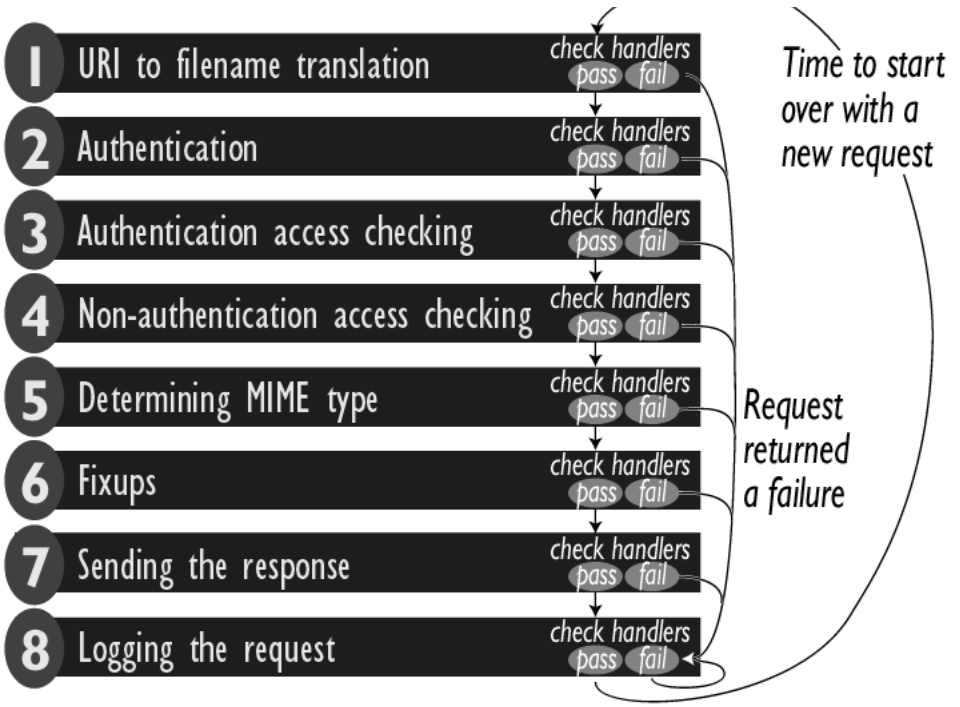


Figure 2.4 Apache handler processing.

Phases in Detail

To give you a better understanding of how Apache works, the next section will explain each of the eight phases in more detail.

Phase 1: URI to File Name Translation

Requests come in with a URI, which is a subset of the URL that identifies the path to the Web page. If you were to open the URL http://www.linuxcertification.com/index.epl in a Web browser, the browser would connect to the host named www.linuxcertification.com and send it the following request:

```
GET /index.epl
```

The path name `/index.epl` is the URI. Apache needs to find out the file on the file system to which this corresponds. It starts at the document root, which is the root of all the server's Web documents, and walks the tree down to the requested file. In this case, if the document root was `/home/httpd/html`, the corresponding file would be `/home/httpd/html/index.epl`.

Sometimes the translation is not as simple as it seems, as Apache has command directives that can be placed in its configuration files that alter the way URI-to-file-name translation works. One of the most common is with the `ScriptAlias` command, used to specify both the URI prefix and the file system location of CGI files. A `ScriptAlias` command looks like this:

```
ScriptAlias /cgi-bin/ /home/httpd/cgi-bin/
```

If Apache received the request for `/cgi-bin/index.epl`, it would have to look in `/home/httpd/ cgi-bin` for the proper `index.epl` file.

Phase 2: Authentication

Authentication is the process of checking the user's credentials (username and password, typically) with one or more lists of credentials Apache keeps. Different modules authenticate in different ways; the module `mod_auth` looks up usernames and passwords in text files, while `mod_auth_db` and `mod_auth_dbm` look them up in database files.

This phase does not actually check to see if the user is authorized to view the requested resource; it just checks to see if the user is who he says he is. In other words, do the username and password presented match the list of known usernames and password combinations?

Phase 3: Authentication Access Checking

It is at this point that Apache compares the authentication information from the previous phase with access rights to the particular resource. The way Apache handles authentication access checking is also tied to the particular module in use; standard practice is to check the `.htaccess` files inside of a particular directory (and all of its parent directories) for this information, but it also can come from database servers or the `httpd.conf` file itself.

Phase 4: Non-Authentication Access Checking

After Apache checks access based on the user's credentials, it checks access based on other factors, such as the requesting IP address. In the `httpd.conf` file, access to certain directories can be denied based on IP address.

Because multiple restrictions can be placed on access checking, all handlers registered for this phase should run.

Phase 5: Determining Mime Type

MIME types let the Web browser know what type of file it is receiving—a plain text file, an HTML file, a JPEG, and so on. The standard modules `mod_mime` and `mod_mime_magic` are used to find out the particular MIME type of a file.

Apache uses the two modules to figure out MIME types in two different ways. `mod_mime` checks for the extension of a file. To `mod_mime`, `.txt` is a text/plain file, `.html` is a text/html file, and so on.

`mod_mime_magic` checks the contents of a file, typically the header, to determine file type. `mod_mime_magic` uses the same "magic number" database that many Linux programs use, including the file command. This type of checking can be more reliable, but it takes more time to perform.

Phase 6: Fixups

Fixups are generic tweaks that Apache makes to a response before sending it to the client. This is a catch-all phase that typically includes parsing server-side Includes, manipulating HTML headers, rewriting URIs, and handling Server-Side Redirects.

The process of the Fixup phase is fairly simple. Typically it is anything that needs to alter the response before it gets sent back to the client.

The Fixup phase runs all of the available handlers registered for it, so that if multiple fixups are needed, all will be processed.

Phase 7: Sending the Response

Sending the response means taking the referenced file, reading it from the file system, optionally processing it (such as by running a PHP script), attaching a header, and sending it back to the client.

Phase 8: Logging the Request

Logging needs to happen whether the response was sent or not. If the request was successful and a response was sent, the log will show an HTTP status of 200 (`OK`). Otherwise, it will show a request with an error code, such as 404 (`Not Found`) or 500 (`Server Error`).

This phase always runs because all requests need to be logged. Additionally, all handlers registered for this phase will run so that Apache can log the request and response in multiple places, if necessary.

The Future: Apache 2.0

As of this writing, Apache 2.0 is in the beta stage. While not yet suitable for stable Web serving, the beta version hints at what will be offered.

Apache 2.0 still has many of the same features you are used to with 1.3.x, including a very familiar configuration file. Most of the changes are under the hood, with a radically improved API, new and improved modules, and IPv6 support.

New Modules

Following is a list of a few recently-developed modules, and descriptions for each:

mod_dav. Supports the HTTP Distributed Authoring and Versioning specification for posting and editing Web content. This module will use Apache to tie Web development and Web serving to HTTP. See www.webdav.org for more information.

`mod_charset_lite`. Currently an experimental module, `mod_charset_lite` allows for character set translation or recoding.

`mod_file_cache`. As the file name hints, `mod_file_cache` caches frequently used static files. The module does its work at server startup, opening the file and saving it into memory. `mod_file_cache` does not work on dynamic content, such as CGI files.

Improved Modules

With Apache 2.0, there will be a number of improved modules for the Apache server. A brief description of three of these, `mod_auth_db`, `mod_auth_digest`, and Filtering, is presented in this section.

`mod_auth_db`. Currently `mod_auth_db` supports the Berkeley DB versions 1.85 and 1.86, with possible extension to DB version 2.0 if you enable compatibility mode. In Apache 2.0 `mod_auth_db` will support DB version 3.0.

`mod_auth_digest`. In Apache 1.3.8 and above, `mod_auth_digest` is an experimental module that uses MD5 Digest Authentication for user authentication. In 2.0, `mod_auth_digest` uses shared memory to offer more support for session caching across processes.

Filtering. It is nearly impossible now to parse content with both server-side includes and CGI scripts. 2.0 will introduce the ability to write Apache modules as filters that will act on the stream of content as it travels to or from the server. Manipulating the stream of content will allow you to massage the data multiple times with different modules.

WARNING Apache 2.0 makes significant changes to the API. Third-party modules that work with version 1.3 will not work with 2.0.

To find out more about Apache 2.0, visit http://httpd.apache.org/docs-2.0/.

Introduction to Logging

Apache has a logging mechanism that allows tracking of what goes right and what goes wrong in Apache's interactions with Web clients. A different log or set of logs can be set for every virtual domain on a system. Each log can be configured to track different aspects of client interaction.

Log formats can be custom configured and named for reuse. Having a set of named log formats allows different virtual servers or different configurations based on the same build to have different log file formats.

The following are examples of some custom logs and their names:

```
HostnameLookups On
ErrorLog /var/apache/logs/error_log
```

```
LogLevel warn
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \ \"%{User-Agent}i\"" combined
LogFormat "%h %l %u %t \"%r\" %>s %b" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent
```

The following is an example of the use of the previously named logs:

```
CustomLog /var/apache/log/access_log common
CustomLog /var/apache/log/referer_log referer
CustomLog /var/apache/log/agent_log agent
CustomLog /var/apache/log/access_log combined
```

Activate `ServerSignature` to add a line to all Apache logs with the server version number and the `ServerName` of the virtual host that generated the line. The default is `Off`.

```
ServerSignature Off
```

The `Alias` command allows for a link from a decoy location to a real location.

Base Systems

Configuring Apache's multiple options can take time, effort, and some frustration. This section, which covers running multiple daemons, easy (and not-so-easy) configuration of Apache, configuration of CGI, and more, will provide a roadmap for Apache administrators who feel the need (or responsibility) to change and tune their Apache installation.

Multiple Daemons

The single daemon method may not always be possible, depending on the needs of the host clients. Multiple daemons can still provide the services needed with a different configuration strategy that spreads across multiple Apache installs. A notable configuration parameter is the `httpd.conf Listen` directive, which defines the IP address for each `httpd` daemon.

When several different daemons start, only the configuration files (and the `log` and `pid` files they point to) need to be different. The same binary

could be used to start each one with an `-f` directive to specify the configuration file.

Take the following considerations into account before running more than one daemon.

Additional resources will be necessary for each site the user hosts as each will have a separate `httpd` daemon that requires system resources. Minimize the impact by using the same binary file for all daemons and specifying different `httpd.conf` files with the `-f` command-line option.

Complexity will increase as each daemon requires a complete Apache directory tree for its hosted site, including `httpd.conf` and possibly other configuration files. Set-up also involves making sure all path references point to the proper trees for each daemon that will run.

Using multiple daemons is preferable when sites require a more secure implementation because the administrator can give each `httpd` process and directory tree access rights that are specific to a hosted site. For example, the single daemon method requires that all Webmasters share a common `httpd.conf` file; the multiple daemon method gives each hosted site its own files. Use of the `Include` directive to bring a different virtual host container for each site into `httpd.conf` could provide similar protection.

Configuration

With the single daemon method, the user can control almost everything within one file, `httpd.conf`. `httpd.conf` is still going to be the focus of the major settings; however, now there is one for each daemon.

The only new ground to cover is the `Listen` directive. Every host that will have its own daemon will require an IP address. The address can be an IP alias or any valid Internet address. Assuming the paths are set correctly (they will be if you installed from source), the rest is essentially the same as a stand-alone server. Table 2.2 is a quick review of the important directives that each host needs.

As with a stand-alone server, Apache must be started with a script that is provided within the Apache directory tree. Each host should have its script invoked before it can serve content. Provide the path to the startup script with the desired option (`start`, `stop`, `restart`). The following has been

Table 2.2 Configuration Directives

`Listen`	Regular or aliased IP address for the host
`ServerRoot`	Base directory of each host's Apache tree, including configuration files and such
`ServerAdmin`	Admin-email@address.com (Error messages from the server and visitor questions/comments will be directed to the specified mailbox.)
`ServerName`	Defines the domain name of the host
`DocumentRoot`	Location for the content of each host
`ErrorLog`	Directory for the error logs
`CustomLog`	The day-to-day logs regarding files served and clients who access the host

taken from a session where both example hosts have been started on our test server:

```
# /www/host1/bin/apachectl start
/www/host1/bin/apachectl start: httpd started
# /www/host2/bin/apachectl start
/www/host2/bin/apachectl start: httpd started
```

Start each host manually as outlined previously or direct the hosts to start at system boot by placing the preceding commands in the preferred startup script. For example, with Red Hat 6.2, the `/etc/rc.d/rc.local` file would be a suitable place.

Running Multiple Daemons

Adding a daemon will require a certain configuration overhead beyond that required for adding a virtual server. Running multiple daemons requires a complete set of directories for each host (including Web pages, logs, and configuration files). All directories could be placed in a site adjacent to each other in the same Apache directory; they could end up in the

home directory of the site's owner. Log files and configuration files could be in the same directories and distinguished by file names. Consider using a separate partition for each user's log files, data files, and content directories so that one user cannot fill up all available space and frustrate the other users. Choice of configuration depends on the needs of the site and the relationship between site owners.

To create the file structure necessary to run multiple daemons, start with the directory tree from an initial Apache installation; duplicate the configuration directory and the initial documents directory, creating a distinctive name for the new server. Use `cp -Rp` to perform the copy recursively and preserve file ownership and attributes. At this point, change ownership of the directories to a different user if necessary. Edit `httpd.conf`, defining different values for items that need to be different. Be sure that the `PidFile` directive and log locations are different and that the `log` directory exists (multiple daemons can share the same log directory if they have different log file names). Do not forget to set the `ServerName` directive to the name of the new server. Usually, there should be different locations for the document directories. Any read-only directories can be shared if appropriate, such as the `icons` location and possibly `cgi-bin`, depending on how closely related the sites are. Certainly, at least one of the `BindAddress` (or Listen) or `Port` directives will have to be edited to differentiate the sites. Read the entire `httpd.conf` file for detailed descriptions of each directive. This will help discover other settings that need to be changed or kept the same on the site.

Set permissions for the contents of each Apache directory tree that suit the needs specific to a given site. This may mean creating specific users and groups for each host or any number of configurations that will be very specific for a given location. The important thing is to determine who needs access to each part of a host's directory tree and then create users, groups, and permissions that reflect those needs. Remember to distinguish between the limited user/group that owns the server process and the real users who will be editing the Web directory tree.

Multiple Daemon Verification

Each daemon can be accessed via its IP address, so the verification method is identical to that which was used for IP-based hosting.

All daemons need to be running; this requires that a startup script be invoked for each daemon.

If problems are encountered, do the following:

- Check if there is an alias for each host.
- Invoke `ifconfig -a` and see if the correct interfaces are present.
- Check that the BindAddress directive has been defined and corresponds to an alias.

Number of httpd Processes

A server process can be run in one of two modes. Many servers are managed by `inetd`, a server process (or daemon) that listens for incoming requests and then starts a new server to match the request. This is the common way to start such lightweight servers as Telnet and FTP.

`inetd` is not practical for larger servers, such as Apache, because they take too much time to start and are called too many times on an active system for `inetd` to be practical. Instead, they run as stand-alone processes (set option `ServerType` in `httpd.conf` to `standalone` rather than `inetd`). This means that a daemon for httpd is started and spawns new copies of itself (servers), which actually handle the incoming requests. Each request is handled individually by one copy. So that it will be ready immediately, the daemon spawns several copies initially (`StartServers`). Each child process can be set to self-destruct after handling a certain number of requests (`MaxRequestsPerChild`) to compensate for memory leaks in some systems, or the child process can be allowed to run indefinitely. If the load increases, the daemon spawns more child processes so that several spares (`MinSpareServers`) stay in reserve. If the load decreases and the number of idle child processes exceeds `MaxSpareServers`, the excess child processes are killed off to free resources. If the load increases a great deal, the excess clients above `MaxClients` are ignored, in order to place a limit on how much of a load Apache places on the system. Monitor the system load and determine if the number of clients is anywhere near `MaxClients`. Be careful that the `MaxClients` number is not too low. If anticipating high initial loads, increase `StartServers` and the other values.

TIP **Use apachect1 fullstatus to help monitor usage.**

Alias

The `Alias` directive is a way of creating the equivalent of symlinks for our URLs. The syntax is:

```
Alias fakename "realname"
```

If `fakename` is `/graphics/` the server will require the trailing slash before it can be found.

For example, `http://localhost/graphics` will return an error.

CGI Scripts

CGI is a stand-alone process that is forked by Apache. CGI can do whatever the server operating system will let it do. It is launched by the user that Apache runs under, but if it has the `suid` or `sgid` set, it will give those permissions, too; therefore, CGI can be a dangerous process.

The worst-case scenario is that `.cgi` is activated for the whole server, and someone creates a `.cgi` file somewhere in the home directory or the Web directory. That person can then pull up that URL in a Web browser and execute the script. But, if the user can create a `.cgi` file in the home directory, then the user can probably create one in the main `cgi-bin` directory as well, and there is the original problem. Regardless, by the time the errant `.cgi` file is found, it is likely that the script will have already run. In the end, limiting `.cgi` to just one directory does not help security significantly, but it may make it easier for the administrator to discover breaches.

The following are a couple of directives for enabling CGI scripts.

ScriptAlias Directive

`ScriptAlias` works just like `Alias`, only it marks the target directory as containing CGI scripts. So, if assuming the following:

```
ScriptAlias /cgi-bin/ /usr/local/my_dir/cgi-bin/
```

then the URL http://localhost/cgi-bin/my_cgi.cgi will execute `my_cgi.cgi` if it exists in:

```
/usr/local/my_dir/cgi-bin
```

AddHandler Directive

`AddHandler` maps a file extension to a given handler. To apply the `AddHandler` CGI script to the extension `.cgi`, type:

```
AddHandler cgi-script .cgi
```

How to Configure CGI

Apache can be asked to enable CGI in two different ways:

- All Web pages referenced within a certain directory are scripts, i.e. `/cgi-bin/`.
- All Web pages with a certain extension are scripts, i.e. `.cgi`.

These two means can be mixed and matched to a fine degree within different directories, but each level of detail adds processing overhead to the Apache server.

Regardless of Apache's configuration, CGI scripts must be executables and properly marked as such (i.e., `chmod +x`). Apache must also be told to reread its configuration file (with the HUP signal) for these changes to take effect.

Enable within Certain Directories

The first and perhaps most common method of enabling CGI is to turn it on for all files in a certain directory. Apache can be told, for example, that all files in the `/home/httpd/cgi-bin` directory are CGI scripts and that these can be executed by calling them with a `/cgi-bin/` prefix. The following line, added to Apache's `httpd.conf` file, accomplishes this:

```
ScriptAlias /cgi-bin/  /usr/local/apache/cgi-bin/
<Directory /usr/local/apache/cgi-bin>
     AllowOverride None
     Options None
     Order allow,deny
     Allow from all
</Directory>
```

Enable within Document Root for Certain Mime Types

A single line added to `httpd.conf` tells Apache that all scripts with a `.cgi` extension are CGI scripts.

```
AddHandler cgi-script .cgi
```

Apache Initialization

When Apache starts up, it begins as root reading the `/etc/apache/httpd.conf` file. It opens up the log files and then launches a number of child processes, as defined by the `StartServers` directive in the `httpd.conf` file. These child processes run as the user and group defined by the `User` and `Group` directives in the `httpd.conf` file. This can be seen by starting the Apache server and typing:

```
root@foo apache $  ps -ef | grep httpd
```

This script returns the following:

```
Owner        PID    PPID
root10185     1   0 09:31:14 ?     0:01 /usr/apache/bin/httpd
www-data 10189 10185  0 09:31:15 ?     0:00 /usr/apache/bin/httpd
www-data 10186 10185  0 09:31:15 ?     0:00 /usr/apache/bin/httpd
www-data 10196 10185  0 09:31:45 ?     0:00 /usr/apache/bin/httpd
www-data 10188 10185  0 09:31:15 ?     0:00 /usr/apache/bin/httpd
www-data 10190 10185  0 09:31:15 ?     0:00 /usr/apache/bin/httpd
```

Process ID `10185` is running as root; this is known as the Apache parent process. The `StartServers` directive is set to `5` in the `httpd.conf` file, and thus we see five other `httpd` processes running as `www-data`. Their parent process ID is `10185`, the Apache parent process.

Changing the `httpd.conf` file makes it necessary to tell Apache to reread it by stopping and restarting the server. This is accomplished by sending Apache a signal asking it to do so. There are two methods for doing this:

`restart.` Restarts every process immediately.

`graceful.` Allows current requests to complete before restarting the server.

A graceful restart is generally preferred, as it avoids disrupting users. You can use the `/usr/apache/ bin/apachectl` tool to do this, as shown here:

```
root@foo apache $  bin/apachectl graceful
```

Following is the returned process listing:

```
Owner         PID    PPID
root10185      1   2 09:31:14 ?     0:01 /usr/apache/bin/httpd
www-data 10230 10185  1 09:41:27 ?     0:00 /usr/apache/bin/httpd
www-data 10233 10185  1 09:41:27 ?     0:00 /usr/apache/bin/httpd
www-data 10231 10185  1 09:41:27 ?     0:00 /usr/apache/bin/httpd
www-data 10232 10185  1 09:41:27 ?     0:00 /usr/apache/bin/httpd
www-data 10229 10185  1 09:41:27 ?     0:00 /usr/apache/bin/httpd
```

The previous child processes have gone away, and the parent process has launched a new set of children.

Modules are called once to initialize themselves and then again for each configuration directive. All of their configuration is built into memory at startup and never changed again. Modules attach their configurations to a tree of directories and locations held in memory, mirroring the `<Directory>` and `<Location>` entries in the configuration file. Modules are also called back as each child process starts and stops.

There are four command-line options to `bin/httpd` that are particularly useful when dealing with modules.

The first is `-t`, which asks Apache to check that the configuration is all right.

```
root@foo apache $  bin/httpd -t
Syntax OK

root@foo apache $
```

This will catch syntax errors and contradictions in the file. The `apachectl` script runs this before trying to restart the Web server, and ensures that it will not shut down the server and be unable to restart it.

The second option, `-l`, will show a list of modules currently compiled into the server.

```
root@foo apache $  bin/httpd -l
Compiled-in modules:
  http_core.c
  mod_env.c
  mod_log_config.c
  mod_mime.c
  mod_negotiation.c
  mod_status.c
  mod_include.c
  mod_autoindex.c
  mod_dir.c
  mod_cgi.c
  mod_asis.c
  mod_imap.c
  mod_actions.c
  mod_userdir.c
  mod_alias.c
  mod_access.c
  mod_auth.c
  mod_so.c
  mod_setenvif.c
root@foo apache $
```

NOTE **The `-l` argument shows only the compiled modules and not the dynamically loaded modules.**

The `-L` option shows the configuration directives they define; the `-h` option shows a short summary of available command-line options.

Log Files

There are a number of different log files ready when Apache starts, as well as ways to configure new log files for an individual Apache install. Utilizing log files, which may seem daunting at first, will provide invaluable information to the log-savvy administrator, whenever anything goes wrong (or right) with an Apache install, configuration, and use.

/var/apache/logs/access_log

Apache adds a line to this log file for every request it receives. By default, entries are similar to the following:

```
192.168.1.1 - - [05/May/2000:12:32:52 -0700] "GET / HTTP/1.0" 200 1622
```

This is known as Common Log Format (CLF).

The fields are as follows:

```
host ident authuser date request status bytes.
```

If no information exists, the log shows a hyphen (-) in its place.

host

This is the IP address the request came from. Apache can automatically do a reverse look-up on each IP and fill in the actual host name here. To enable this, edit the `httpd.conf` file and turn on the `HostnameLookups` directive. Type:

```
HostnameLookups On
```

DNS look-ups tend to be very slow, so this feature should not be activated. If actual host names must be tracked for traffic analysis purposes, do the DNS look-ups at log analysis time. Most popular log analysis tools have an option to resolve IP addresses.

ident

Apache can do an `identd` check (see RFC 1413) on the incoming HTTP connection. An `identd` check is an attempt to determine the identity of a user of a particular TCP connection. To turn on this feature, edit the `httpd.conf` file, and add:

```
IdentityCheck On
```

`IdentityCheck` requires that the client machine run `identd` if it is to work.

NOTE **`identd` checks can be extremely slow, so do not use this feature on a busy Web server. `identd` checks can also be faked, so the information should not be explicitly trusted.**

authuser

If the requested document requires access authentication, then the authenticated user ID is placed in this field.

date

This field contains the date and time the request was initiated. The format is as follows:

```
05/May/2000:12:08:19 -0700
```

request

The actual HTTP request that the client sent is enclosed in quotes.

status

This field contains the three-digit HTTP status code returned to the client from Apache. The most common codes are 200, 202, 301, 302, 400, 403, and 404.

bytes

This is the number of bytes in the response sent back to the client, not including the response headers.

Log File Formats

The actual log file format can be changed. Most log analysis tools understand the default CLF format, but they may also support other formats. Apache allows customization of log file formats using the `LogFormat` and `CustomLog httpd.conf` directives.

To use the CLF format, type the following in the `httpd.conf` file:

```
LogFormat "%h %l %u %t \"%r\" %>s %b" common
CustomLog /var/apache/logs/access_log common
```

The `LogFormat` directive defines a format and assigns it a name. In this case, it is named `common`. Then the `CustomLog` directive tells Apache that the `/var/apache/logs/access_log` file is in the `common` format.

Another popular format is the combined log format, which includes the referring URL and the browser type in each request.

To use the combined log format, modify the `LogFormat` line:

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined
```

Design Your Own Log Files Format

Table 2.3 shows log format modifiers. The listed commands can be used to customize log format files.

http://httpd.apache.org/docs/mod/mod_log_config.html

The `'...'` portion in each of these arguments can be nothing, as in `%a`, or it can be a list of response HTTP status codes on which this modifier should be invoked. For example:

```
%400,501{User-agent}i
```

This would log the type of browser that sent a strange HTTP request. If the HTTP status code for the request is not 400 or 501, a hyphen (-) is placed in the log file in this modifier's position.

/var/apache/logs/error_log

This file contains a log of Apache errors. A typical line that is always logged on server startup is this one:

```
[Fri May  5 12:08:08 2000] [notice] Apache/1.3.14 (Unix) mod_perl/1.21 configured --
resuming normal operations
```

Once the server restarts, the configured logging level takes effect. This logging level is set using the `LogLevel` directive in the `httpd.conf` file:

```
LogLevel warn
```

Table 2.4 gives the meaning of various log levels.

For example, when the log level is set to `warn`, all message types from the `warn` level and above are shown. Setting the log level to `warn` means that `emerg`, `alert`, `crit`, and `error` messages are also sent.

Table 2.3 Available Log Format Modifiers

COMMAND	DESCRIPTION
%ºa	Remote IP address.
%ºA	Local IP address.
%ºB	Bytes sent excluding HTTP headers.
%ºb	Bytes sent excluding HTTP headers. This modifier is in CLF format, which means you see a "-" rather than a 0 when no bytes are sent
%º{FOOBAR}e	The value of the environment variable named FOOBAR.
%ºf	File name.
%ºh	Remote host.
%ºH	Request protocol.
%º{foobar}i	The contents of Foobar's header line(s) in the request sent to the server.
%ºl	Remote logname from identd if supplied.
%ºm	Request method.
%º{Foobar}n	The contents of note "Foobar" from another Apache module.
%º{Foobar}o	Contents of Foobar: header line(s) in the server's reply.
%ºp	Canonical port of the server serving the request.
%ºP	Process ID of the child that serviced the request.
%ºq	The query string (prepended with a ? if a query string exists; otherwise, an empty string).
%ºr	First line of request.
%ºs	Status. For requests that got internally redirected, this is the status of the original request – use %…>s for the last request.
%ºt	Time in common log format time, which is standard English format.
%º{format}t	Time in the form given by format, which should be in strftime(3) format; potentially localized.
%ºT	The time taken to serve the request in seconds.
%ºu	Remote user (from auth); note it may be bogus if return status (%s) is 401.
%ºU	URL path the client requested.
%ºv	Canonical ServerName of the server serving the request.
%ºV	The server name according to the UseCanonicalName setting.

http://httpd.apache.org/docs/mod/mod_log_config.html

Table 2.4 Logging Levels––Meanings and Examples

LOG LEVEL	DESCRIPTION	MESSAGE DISPLAYED
`emerg`	Emergencies—system is unusable	"Child cannot open lock file. Exiting."
`alert`	Action must be taken immediately	"getpwuid: couldn't determine user name from uid"
`crit`	Critical conditions	"socket: Failed to get a socket exiting child"
`error`	Error conditions	"Premature end of script headers"
`warn`	Warning conditions	"(child process 1234 did not exit) sending another SIGHUP"
`notice`	Normal but significant conditions	"httpd: caught SIGBUS attempting to dump core in…"
`info`	Informational	"Server seems busy (you may need to increase StartServersor Min/MaxSpareServer)…"
`debug`	Debug-level messages	"Opening config file…"

http://apache.org/docs/mod/core.html

Shells and Commands

Almost any complex program has smaller programs that will run alongside it to provide information about how well (or poorly) the program is running, which resources the program is using, who is using the program, and so forth. The following sections will discuss a few programs to help the Apache administrator with tuning their Apache install.

Benchmarking

Benchmarking is the art of running a program and testing the program for faults, bugs, speed, efficiency, and more. In this section, benchmarking programs will be discussed, and some examples will be given of useful benchmarking scenarios and programs.

ab

Apache comes with a benchmarking program called `ab` (Apache bench). It will be in `bin` within the main Apache directory if it has been installed from source. Although it is difficult to simulate a real-world situation, `ab` can put a heavy load on Apache and the underlying Linux system.

The following is the syntax for `ab`:

```
ab options  http://hostname:port/path
```

It ends up looking something like this in operation:

```
# ab -n1000 -c100 http://192.168.1.1:80/htdocs
```

A number of options allow benchmarks to be tailored to run in various ways. Some of the main options for use with `ab` are:

- **`-n` (number of requests).** This specifies how many times the specified page should be accessed. It can provide information about the speed of the server machine and the responsiveness of Apache.
- **`-c` (concurrency).** This option simulates a number of simultaneous connections. It defaults to 1, but 1,000 or more can be defined. Setting the `-c` and the `-n` options to high numbers can give an approximation of how the server works under a high load.
- **`-v` (verbose).** This option prints more troubleshooting information.

Running a benchmark with `ab` involves starting a server, determining what page to test, and then invoking the benchmark with the desired options.

The following example is one of the larger pages in the Apache documentation that comes with the server:

```
# ab -n1000 -c100  \ http://192.168.1.1/htdocs/manual/directives.html
Server Hostname:        192.168.0.2
Server Port:            80

Document Path:          /manual/directives.html
Document Length:        12098 bytes

Concurrency Level:      100
Time taken for tests:   21.435 seconds
Complete requests:      1000
Failed requests:        0
```

```
Total transferred:      12346000 bytes
HTML transferred:       12098000 bytes
Requests per second:    46.65
Transfer rate:          575.97 kb/s received
```

This benchmark was run on a 486 DX2/80 with 64MB of RAM. That would be an impressive performance if it were actually serving content at that rate on the Internet. The test was performed locally without any of the overhead that might occur when running over a slow connection. The results are further skewed by the fact that `ab` itself can consume a chunk of the system resources as it tests the server.

Remember these are rough approximations of what happens in a real-world situation.

With the benchmark, it may be useful to find out how many connections it takes before the machine starts swapping. Then add more RAM accordingly. If a user is running dynamic content and needs to check how well a new script performs under load, this may indicate if it could become a bottleneck.

System Utilities

Some system utilities will help the Apache administrator perform his or her job well, with a minimum of headaches and complications. Utilities exist for helping a user (or administrator) write CGI scripts, analyze logs, and test Apache performance. This section presents an array of different utilities available for the Apache administrator.

Creating CGI Scripts

After configuring for CGI and reloading, Apache is ready to start serving CGI scripts. Several standard CGI scripts come with Apache, usually contained in the `/home/httpd/cgi-bin` directory.

Content of CGI Output

One of the most important things to remember when writing CGI scripts is that they must produce not only the content of the Web page, but the headers as well. Although a Web page can be as simple as the following:

```
<html><head>
<title>My Web Page</title></head>
 <body><h1>My Web Page</h1></body>
</html>
```

the Web client really receives this:

```
HTTP/1.1 200 OK Date: Wed, 20 Dec 2000 23:41:38 GMT Server: Apache/1.3.14 (Unix)
mod_perl/1.21 PHP/4.0.11 X-Powered-By: PHP/4.0.11 Connection: close Content-Type:
text/html
<html><head>
<title>My Web Page</title></head>
<body><h1>My Web Page</h1></body>
</html>
```

Apache typically adds the header, which consists of all the lines of text before the HTML, when it serves a static Web page. CGI scripts must provide their own headers. At minimum, they must provide a `Content-Type` line. A CGI script that wants to produce a simple Hello World! must actually produce:

```
Content-type: text/plain
Hello World!
```

Failure to add proper headers is a common mistake of new CGI programmers. Also, there absolutely must be a blank line between the `Content-Type` line and the Hello World! line.

Example CGI Scripts

The following scripts demonstrate some of the basics of CGI programming. They progress from extremely simple to mildly involved. Although they use the bash shell to serve their content, they could be written in any language that Linux executes, as long as the scripts produce exactly the same output.

```
Hello World!
```

The following shell script produces a simple `Hello World!` on the screen:

```
#!/bin/sh
# helloworld.txt.sh -- write Hello World as a CGI script, in text.
echo Content-type: text/plain
echo
echo Hello, World!
```

Because the `Content-Type` was text/plain, the script was produced in plain text, as shown in Figure 2.5. A common scripting mistake is to forget to leave NO space after the end of the text/plain line and have the second echo on a separate line.

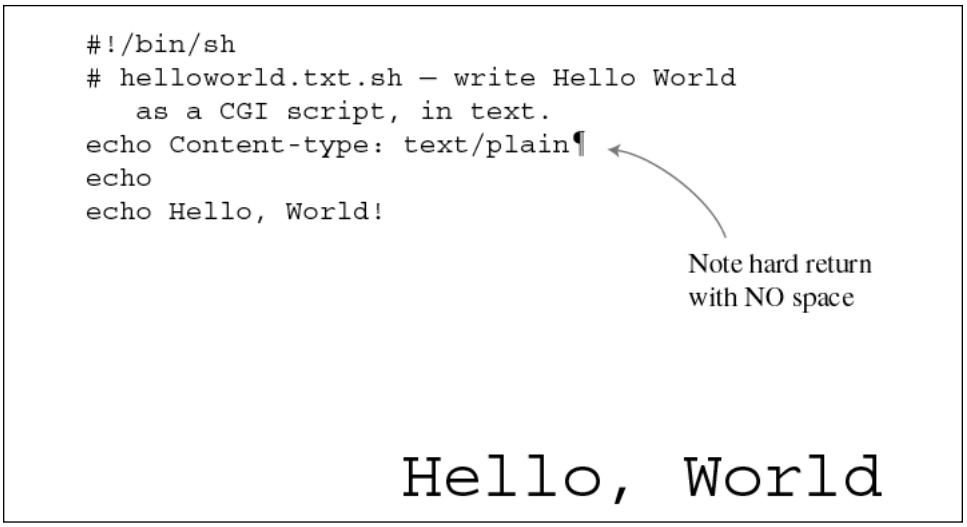


Figure 2.5 A simple, plain-text "Hello, World!" CGI script in plain text.

Here is the same script, changed to HTML and with markup added (see Figure 2.6):

```
#!/bin/sh
# helloworld.html.sh -- write Hello World as a CGI script, in HTML.
echo content-type: text/html
echo '<head><title>Hello, World!</title></head>'
echo '<body bgcolor="#FFFFFF"><h1>Hello, World!</h1></body></html>'
```

NOTE **The quotes around the text in this example are there for a reason: the comma (,) and exclamation point (!) characters are reserved characters in shell parlance and would cause confusion if they were not quoted.**

Server Environment

Just as a normal process inherits the environment from its parent, a shell script inherits the environment from Apache. It also receives several other variables. Use the `set` command to show this:

```
#!/bin/sh
# set.sh - show the environment in a CGI script.
echo Content-type: text/plain
echo
set
```

```
#!/bin/sh
# helloworld.html.sh — write Hello
    World as a CGI script, in HTML.
echo Content-type: text/html
echo
echo '<html>
<head><title>Hello, World!</title></head>'
echo '<body bgcolor="#FFFFFF">
    <h1>Hello, World!</h1></body></html>'
```

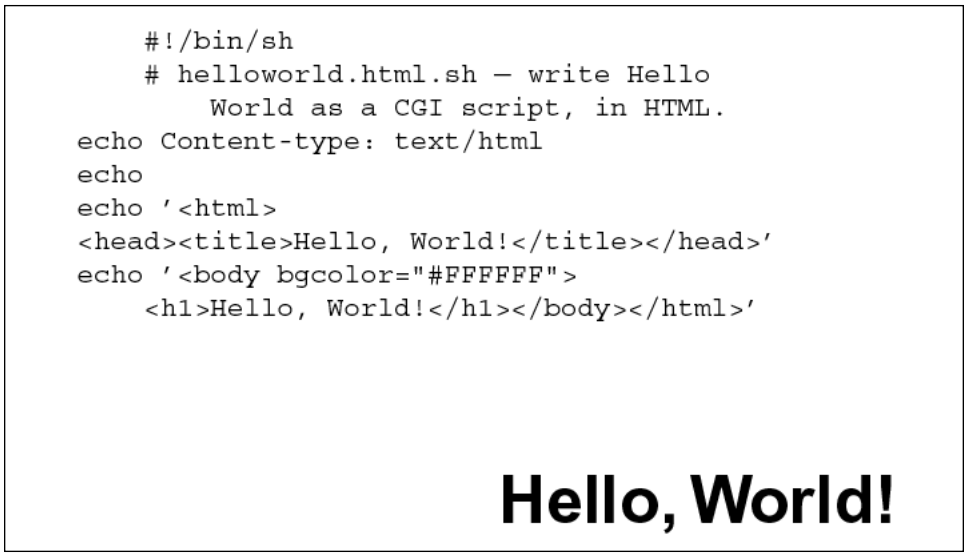


Figure 2.6 A simple, HTML "Hello, World!" CGI script in HTML.

Many of these variables are useful in determining what kind of output is sent back to the browser. This information is also very useful for a cracker trying to break into a Web server; consequently, this is not a script to leave on a Web server once it goes into production.

Form Handler

Many times a CGI script needs input data from the browser to determine what type of information to send back; for instance, a Web mail client would need to know the username of the person for which it displays mail.

Two common methods of passing data to CGI forms exist: the GET method and the POST method.

The GET method sends data after a question mark on the request URI, such as www.test.com/cgi-bin/test?abcd. The script named `test`, in the `cgi-bin` directory of www.test.com was given the data `abcd`.

The GET method is used in HTML input forms, when the type is not specified or when the type is GET. Because the information is passed in the URL, it can be keyed into a browser by hand, without accessing a Web page with an `<INPUT>` tag. The GET method also typically has a limit of 1,024 characters, and the data is URL encoded (spaces are changed to pluses,

some characters are changed to their hexadecimal representation, and so on). The data is entered in the variable `QUERY_STRING`, which becomes part of the CGI script's environment.

The POST method, on the other hand, gives data directly to the Web server. The URL requested by the browser does not have a question mark (?) appended to it, nor is the data sent limited to 1,024 characters. The data is encoded in URL form, but Apache decodes it before sending it to the CGI program. The CGI script must read the `CONTENT_LENGTH` environment variable to determine the length of the incoming data, as no EOF is sent on this stream.

Consider the following CGI script:

```
#!/bin/sh
# hi.sh - say hi to the CGI script
echo Content-type: text/plain
echo
if [ "$QUERY_STRING" = "hi" ]; then
      echo You said hi!
else
      echo You did not say hi!
      echo Please say hi.
fi
```

When called with a `$QUERY_STRING` of `"hi"`, it will respond with `You said hi!` (See Figure 2.7.) Otherwise, it will respond with `You did not say hi. Please say hi.` (See Figure 2.8.)

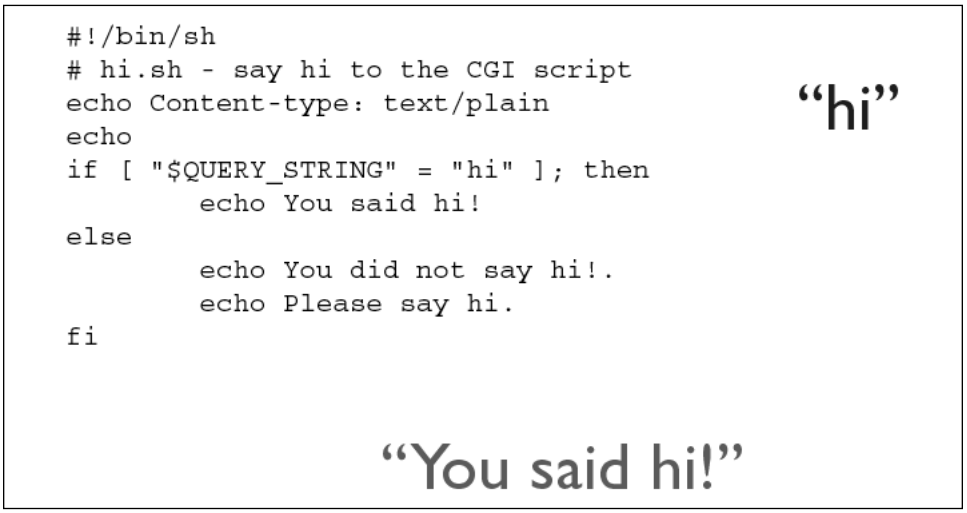


Figure 2.7 Screenshot of CGI script.

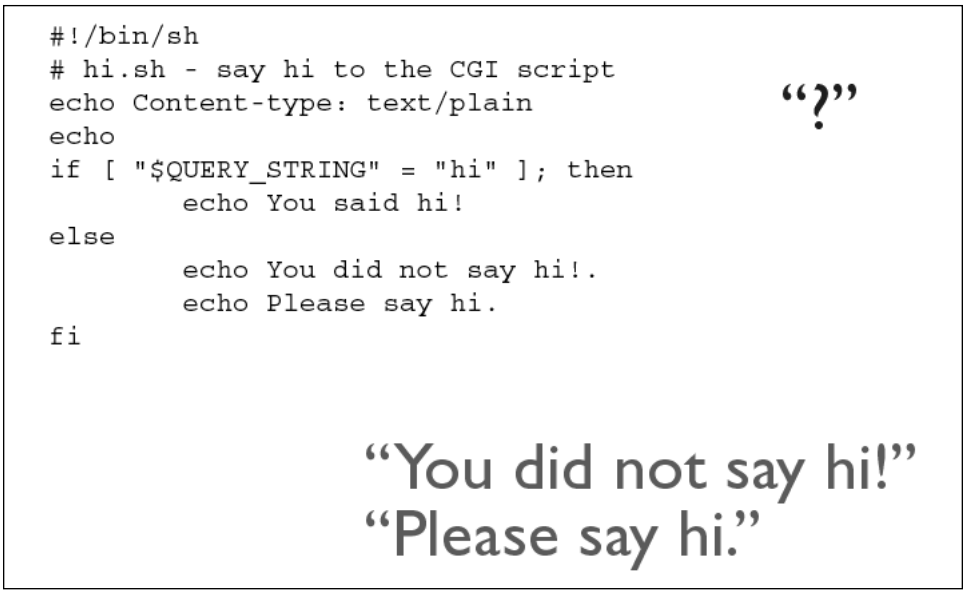


Figure 2.8 Screenshot of CGI script without a $QUERY_STRING of "hi".

Basic Redirect

Because a CGI script controls the header, it is able to do some things that are more difficult or impossible to do with just HTML. One common use of a CGI script is to redirect a user to a different page on a Web server using the Location header.

The Location header can be used to give the client a different page to view. It can be either another document on the same server or a different URL (referencing a different server) altogether.

In the first example, a Web browser requests an old URL. A CGI script gives the new, moved page back to the client. The server acts as if the client had requested the new script, and the client never knows the difference because it still holds the old URL in its window.

```
#!/bin/sh
# old2new.sh - redirect from old CGI script to new CGI script
# redirect to http://my.server.com/cgi-bin/new.sh
echo Location: /cgi-bin/new.sh
echo
```

The second example is a redirect to another server. In this case, the Web browser does get a response from the original server but is told to immediately view a page on a different server:

```
#!/bin/sh
# old2new2.sh - redirect from old server to new server via CGI
# redirect to http://someone.elses.server.com/cgi-bin/new.sh
echo Location: http://someone.elses.server.com/cgi-bin/new.sh
echo
```

A blank `echo` command is necessary even when the header contains a simple location directive.

While Apache's CGI implementation knows the difference between a local and remote redirect, the client browser does not. When working in a non-CGI environment, such as PHP, or with a different Web server, however, do not rely on the server passing the right header.

The basic redirect shown previously does not give the user anything in the browser window while the redirect is taking place. It is possible to return a page, give the user a few seconds to read it, and then redirect the user to a new page. This is through a Netscape Refresh HTTP header extension.

As a Netscape extension, this method is widely used but not accepted by the World Wide Web Consortium. Although most Web browsers will behave properly when seeing the Netscape Refresh HTTP header, some may not redirect but will stay on the original page.

A Refresh tag is similar to the following:

```
Refresh: 5;URL=http://my.new.server.com/
```

In this case, the Web browser loads http://my.new.server.com/ after five seconds. The length can be any number of seconds desired. Here is an example:

```
#!/bin/sh
# redirect.sh - redirect via CGI script a new server
echo Content-type: text/plain
echo "Refresh: 5;URL=http://my.new.server.com/"
echo
echo redirecting in 5 seconds...
```

NOTE **Refresh line was quoted to avoid letting the shell use the semicolon as a command separator.**

Performance Monitoring

Apache comes with a pair of tools in module form that can keep track of the server's performance and show the server's configuration.

mod_status

The first tool, called `mod_status`, can provide a Web-accessible method for seeing what the server is doing.

To enable `mod_status`, load the module and give it scope in `httpd.conf`:

```
LoadModule status_module        modules/mod_status.so
...
AddModule mod_status.c
...
ExtendedStatus On
<Location /server-status>
      SetHandler server-status
      Order deny,allow
      Deny from all
      Allow from localhost
</Location>
```

This set of directives creates a Web page (at the location specified by the `Location` tag) that contains information about the server. It may be a good idea to make this password protected to prevent the rest of the world from monitoring the server's status. In the preceding directives, access is allowed only to the server status from local machines. The following is the output of `mod_status` generated during a benchmarking test:

```
Apache Server Status for localhost.localdomain

Server Version: Apache/1.3.14 (Unix)
Server Built: Dec 17 2000 21:48:11

Current Time: Wednesday, 26-Dec-2000 11:44:46 CST
Restart Time: Wednesday, 26-Dec-2000 11:39:41 CST
Parent Server Generation: 1
```

```
Server uptime: 5 minutes 5 seconds
Total accesses: 2993 - Total Traffic: 35.0 MB
CPU Usage: u20 s40.79 cu0 cs0 - 19.9% CPU load
9.81 requests/sec - 117.5 kB/second - 12.0 kB/request
131 requests currently being processed, 7 idle servers
```

NOTE **ExtendedStatus adds more load to Apache, so unless you are very curious about performance, it is best to leave ExtendedStatus off.**

Using `mod_status` is a convenient way to monitor server performance, especially if there are several that need to be watched. Bookmark them, and check as needed.

NOTE **If mod_status is loaded (or compiled into the server), these directives can be issued from any directory through an .htaccess file.**

Because the output of `mod_status` may contain information that should not be available to the public, its location should be protected. One possible way is as follows:

```
<Location /server-status>
      SetHandler server-status
      AuthType Basic
      AuthUserFile /etc/.htpasswd-status
      Require Valid-user
</Location>
```

This configuration would restrict the page from being seen by anyone who does not have a valid username and password, as is defined in the `/etc/.htpassed-status` file.

Getting the Most Out of mod_status with ExtendedStatus

When using `mod_status` and server status, more information can be obtained by using the `ExtendedStatus` directive. This directive takes a value of either `On` or `Off`. Having `ExtendedStatus` turned on results in a degradation in performance because the server is using resources for capturing the data. Unless actively examining the server's performance and capacity, the user should leave this option off.

server-info

For real-time updates on how a server is configured, the server-info handler supplied in `mod_info` generates a page that reports on the server's configuration. Instead of showing performance data, the page displays information about how the server is configured: what the loaded modules are, what directives they support, and the actual settings of the directives (under some circumstances). Like the information from server-status, this report should be secured from the public eye.

Add these directives to `http.conf`:

```
LoadModule info_module        modules/mod_info.so
...
AddModule mod_info.c
...
<Location /server-info>
    SetHandler server-info
    Order deny,allow
    Deny from all
    Allow from localhost
</Location>
```

Similar to `mod_status`, `mod_info` creates a Web page (at the location specified by the `Location` tag) that shows a detailed listing of the installation, including all installed modules and configuration file directives. Because this module reads server information at run time, any changes made to the configuration files since the last server restart will not show.

Disable modules that will not be used. This will make your computer much faster.

TIP **It is a good idea to restrict access to the server-info page for security reasons. Remember that if mod_info is compiled into the server, it can be used from any configuration file, including the per-directory .htaccess files. You may want to load the module dynamically, making it available only when it is needed.**

Some Good Log Analysis Tools

Once the logs are configured, the user may wonder how to read them. There are several tools available to help view and interpret Apache logs.

Webalizer

Webalizer (www.mrunix.net/webalizer) is a quick, free, multilingual analyzer. It supports CLF, as well as variations of Combined Logfile Format, wu-ftp xferlog, and squid log formats. Configuration happens from the command line or from configuration files.

Analog

Analog (www.analog.cx) is a free, Web-configurable logfile analyzer that is available on many platforms and in many languages. Analog has many supporting tools available to affect output, including Report Magic for Analog (www.reportmagic.com).

Wusage

Wusage (www.boutell.com/wusage) is a shareware log analyzer available for many platforms. It exhibits optional Web-based administration, among other features.

Summary

Summary (http://summary.net) is another piece of shareware. It is available on multiple platforms and supports multiple log file formats.

logresolve

There is also a simple script that comes with Apache called `logresolve`. To turn IP addresses in an `access_log` file into resolved host names, simply run it as shown:

```
logresolve <access_log >access_log.new
```

Understanding how networks function is important for optimizing your Apache server. The next chapter will discuss Apache's relationship with network protocols.

CHAPTER 3

Networking

Objectives

- Define Apache's network protocols.
- List and define Apache's Internet-related features.
- List the Apache Web site hosting methods.
- Define virtual hosting.
- Define the `mod_rewrite` module.
- List the `mod_rewrite` associated variables.

Theory of Operation

In this section of the networking chapter, Apache's use of TCP/IP, HTTP, and other protocols will be discussed. These are important for any Apache administrator to understand because of the inherent purpose of Apache: to serve data over a network.

What Is TCP/IP and How Does Apache Use It?

In a Transmission Control Protocol/Internet Protocol (TCP/IP) network, every computer gets a unique IP address. TCP/IP is the main protocol suite that allows all the computers on the Internet to communicate.

Rather than remembering the number of every computer you want to connect to on the Internet, an easy-to-remember name is used to connect to Web sites. This name-to-number translation is possible with the Domain Name System (DNS), a group of distributed servers that exist to translate the string of numbers (such as IP address 64.208.42.41) into names (such as www.apache.org). Every time the name of a Web site is entered, the name must be resolved to an IP address through a look-up on a DNS server to determine where to send the request.

Apache relies on TCP/IP for many things, not the least of which is to communicate with Internet browsers. Web browsers and servers use a different language that primarily depends on TCP/IP as a transport. Although other transport protocols can be used, TCP/IP port 80 is typically used for Internet Web servers to carry the request/response protocol known as HTTP.

HTTP

Hyper Text Transfer Protocol (HTTP) is the protocol that browsers and servers use to submit and serve requests. Uniform Resource Identifier (URI), Uniform Resource Location (URL), and Uniform Resource Name (URN) are all part of the communication between clients and servers using HTTP to communicate. Each request/response is preceded by a bit of information about the request/response that is contained in a header.

Headers

The client and the server know they are actually dealing with hypertext when they transfer information back and forth because, when they send data, they start their transmissions with headers.

A header is a piece of information that tells the client or the server what kind of information it is receiving. Thanks to headers, the Web browser usually knows the difference between text files, HTML files, JPEG images, and so forth.

Apache makes significant use of headers for everything from file translation to Web browser redirection. Because headers indicate what kind of Web browser is hitting the Web site, Apache can do specific things based on the browser information contained in the header. Headers will be discussed in greater detail later in this book.

What Is HTTP?

Hyper Text Transfer Protocol (HTTP) is an acronym that makes more sense when broken down into its components. Hypertext is the format of the content viewed in a Web browser. The following is an example of a hypertext document.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
        "http://www.w3.org/TR/1999/REC-html401-19991224/loose.dtd">
<html lang="en">
<head>
      <meta http-equiv="content-type" content="text/html; charset=iso-8859-1">
      <title>Untitled</title>
</head>
<body bgcolor="#FFFFFF">

<h2 align="center">
      Welcome to hypertext. Enjoy your stay.
</h2>

</body>
</html>
```

The transfer protocol is the means by which Web servers and clients share hypertext. There is currently a revolution developing in HTTP. HTTP/1.1 is the version that is slowly taking over the Web; it will be discussed in further detail in the next section.

When a Web browser (a client) is directed at a Web site (a server), it makes a request for information. The server receives that request and through a series of phases decides what to do with it. If the server feels inclined, it will send a response back to the client, which the client then renders as content.

HTTP/1.1

HTTP/1.1 has many features above the 1.0 standard. Web browsers are beginning to support the new protocol. Apache is in full compliance with

HTTP/1.1, although it does not implement all features. Luckily there are modules to make up for many HTTP/1.1 features that Apache does not support at its core.

BrowserMatch

Depending on the Web browser hitting a site, Apache can react in different ways, using the `BrowserMatch` directive. `BrowserMatch` goes beyond clients like Netscape and Opera to include clients such as `RealPlayer`. For example:

```
BrowserMatch "Mozilla/2" nokeepalive
BrowserMatch "MSIE 4\.0b2;" nokeepalive downgrade-1.0 force-response-1.0
BrowserMatch "RealPlayer 4\.0" force-response-1.0
BrowserMatch "Java/1\.0" force-response-1.0
BrowserMatch "JDK/1\.0" force-response-1.0
```

Expirations

`mod_expires` is a module that allows Apache to dictate when it should refresh a cached document. Expirations are useful to make sure the client is not viewing cached data when there is a more recent version available. Expirations also serve the purpose of caching information that will not change for a long time, such as images. `mod_expires` controls the setting of the Expires header in server responses to control a client's use of cached files.

Keep-Alive

Keep-Alive is actually an HTTP/1.0 term for what is called persistent connections in 1.1. Persistent connections create the possibility of multiple requests to occur over one TCP connection. Persistent connections can reduce latency in serving HTML documents quite substantially, especially on Web sites with many images. Some Web browsers support Keep-Alive, and others (like Internet Explorer 4.0b2) have a broken implementation.

To support Keep-Alive, turn it on in `httpd.conf` with the following command:

```
KeepAlive On
```

For more information visit http://httpd.apache.org/docs/mod/core.html#keepalive.

Host Header Request

Unlike HTTP/1.0, HTTP/1.1 requires a host header in the client's request, even if it is blank. The following is an example of a 1.1 header that includes the host header:

```
GET /~e8926506/siberia.htm HTTP/1.1
Host: stud1.tuwien.ac.at
```

Chunking

HTTP/1.1 chunks data as it sends it back to the client who requested it. According to HTTP/1.1's RFC, chunking means the following:

> The...encoding modifies the body of a message in order to transfer it as a series of chunks, each with its own size indicator, followed by an OPTIONAL trailer containing entity-header fields.

Further, chunking "allows dynamically produced content to be transferred along with the information necessary for the recipient to verify that it has received the full message."

So what does that mean? Chunking came about as a way to get around the need for a server to know the size of a response it is sending. If a Web server is to use persistent connections with HTTP/1.1, it must know the Content-Length of a response it sends back to a client. Because of the dynamic content created by CGIs and the like, Content-Length of a given page became variable and impossible to know before hand. Chunking was the answer. The server grabs a small bit of the dynamic output, determines its length, and sends it out to the client. Then it goes back for more.

As an additional advantage, once the server has sent all of a chunk-encoded document, it has the option to send additional response headers. Because the server is chunking all the data anyway, these headers can be dynamic, just like the content.

Multiple Hosts

Apache has three separate ways to host many sites on a single machine:

- Run multiple instances of the `httpd` daemon.
- Use Apache's IP-based virtual hosting.
- Use Apache's name-based virtual hosting.

These three methods can be used in any combination, but they are subject to limitations of the operating system. We will examine each method and offer some insight that will assist in choosing what will work best to fulfill a location's hosting requirements. The choice of methods may be limited by facilities provided by the operating system and hardware of the host machine. In the instance of name-based virtual hosting, the methods may be limited depending on whether the intended audience is expected to be using earlier browsers.

The term *virtual hosting* refers to the process of running more than one Web site from a single instance of the Apache Web server daemon. One machine can appear to be many from the perspective of the Internet; hence, the hosts contained within are virtual. The virtual host area is a characteristic of Apache, independent of the host operating system.

At the other end of the spectrum is the extremely busy Web site that requires several machines to handle its load; many machines appear to be one site. This is beyond the scope of the course, as it entails more than just Apache. For most sites that require multiple Web servers, it is best to divide the pages into different sections. Each section should be on a different server, each with a different host name such as www.apache.org, dev.apache.org, and java.apache.org.

The obvious way to run a Web site is to have one machine for each server. This arrangement is easy to describe and relatively simple to implement. The first few machines are not difficult. The limited range of sizes of machines available for any given Web site and any given machine causes the machine to be too large and sometimes even too small for the Web site. For moderately sized Web sites, a typical computer could be several times too large. The benefit of sharing the cost of a computer (including hardware costs, rack space rent, and maintenance costs) among several Web sites becomes obvious. Allocation of excess capacity on a shared server can provide a comfortable safety margin for all Web sites hosted from the server; they are not all likely to have peak usage at the same time.

Base Systems

There are many configuration options for the networking end of Apache. Whether to use multiple instances of Apache for multiple domains, how many Apache daemons should be running, and how to employ IP-based or name-based configurations are decisions that many Apache administrators

will encounter when running Apache. This section will provide information and background that is sure to help administrators as they decide how to use their Apache installation.

Virtual Hosting

When hosting multiple Web sites, users have the option to run a single stand-alone daemon that will handle all of the hosted sites as virtual sites or to have a separate daemon for each site. A combination of these methods will work as well (multiple stand-alone daemons with multiple virtual servers on each one). The single daemon method consumes fewer system resources because there is only one master process running.

When a separate daemon is used for each hosted site, each one will consume a share of system resources, but maintenance will be easier. Moving one of the daemons to a different machine involves only copying it, shutting it down on the first machine, then bringing up the new copy on the new machine.

Running multiple daemons allows each one to run under a separate (restricted) user ID, simplifying security management. This eliminates the need for `suExec` and secures scripts that cannot use `suExec`, such as scripts running under `mod_perl` and `mod_php`.

Running multiple daemons on a machine requires additional resources. Multiple daemons, such as `HTTPD`, would require more memory, more CPU time, and additional IP addresses for each daemon. If the machine does not have the resources to support such demands, consider some of the other virtual hosting options instead.

Configuring Separate Daemons

To run separate instances of Apache on the same server, a different IP address is required for each server. Then tell Apache which IP addresses to listen to with the `Listen` directive. The following is the syntax for `Listen`:

```
Listen <IP Address>:<port>
```

The `Listen` directive tells Apache which IP addresses and port number to listen to for HTTP requests. Each instance of the server should listen to a different IP address:

```
# Listen 123.456.789.10:11
```

If a server configuration does not specify which IP address to listen to, Apache listens to all valid addresses. Hence, when running multiple daemons on the same machine, it is important to specify which address each daemon should listen to.

Directing the Request to a Virtual Host

The first step in sharing a machine among multiple Web sites is finding a means of attaching the Web server to a unique address, which enables a Web browser and the Web server to find each other. With a system using Ethernet, this involves plugging another Ethernet card into an expansion slot and assigning a unique IP address to this new interface. Start another Web server daemon, and configure it to respond to this new IP address. This setup works well and has few limitations.

Although adding more Ethernet cards is not overly expensive, the cost of adding a card for each site may not be justified. The bandwidth of a single card exceeds the needs of most Web sites by such a margin that the first card would have been able to handle several Web sites if only there was a way to share among them. As the user adds more sites to a system, the number of expansion slots could be depleted before the machine's capabilities to run additional Web server daemons are exceeded.

Even though there may be times when the user finds the need to add cards to a machine, obtaining a Web server daemon to distinguish between requests for several different Web sites could allow several different sites to be served through a single card on a single machine.

There are two ways to distinguish between requests, IP-based virtual hosting and name-based virtual hosting.

Single Daemon/Virtual Hosting

This method uses a single instance of the `httpd` daemon in conjunction with IP- or name-based hosting to manage things. Consider the basic characteristics of the single daemon method to determine if it is suitable for a location's needs:

Using the single-daemon `httpd` method has the advantage of consuming fewer system resources because there is only one `httpd` daemon running. On a busy site, this can make a difference in system load.

Settings may be easier to maintain because some of the configurations are shared, such as having only one `httpd.conf` file to edit when making changes to the daemon's behavior. The user can add new hosts quickly by adding a small section in `httpd.conf` with host-specific information. All other major settings will be inherited from the directives outside of the `VirtualHost` sections. Using the `Include` directive can allow creation of separate configuration files for each virtual host.

A potential security/privacy problem arises when using a single daemon. The `httpd` daemon runs with user permissions that have been assigned to it. If Webmasters are to be able to access certain content or tailor the server's behavior to their needs, they must have the permissions to implement them. Permissions also give them the ability to see or even change parameters that control the other hosts.

IP-Based Virtual Hosting

If there are many available IP addresses, IP-based virtual hosting is one option for hosting multiple Web sites. Each host has a unique network address that allows it to be found on the network. IP-based virtual hosting is operating system dependent and requires IP aliasing, the assignment of multiple IP addresses to the same interface.

To set a single NIC up with multiple IP addresses with IP aliases, enable the feature in the kernel.

IP aliasing should be enabled by default with most distributions. If IP aliasing has not been enabled, the user will need to recompile the kernel to enable support for this feature. IP aliasing can be found under `Network Options` in the kernel configuration menus.

Use the `ifconfig` (interface configuration) command to control which IP addresses are attached to the interfaces on a system. Initially an interface has a single IP address, determined when the system is first configured.

Issuing the `ifconfig` command with no arguments displays the current status of all interfaces on the system:

```
# ifconfig
eth0     Link encap:Ethernet HWaddr 00:80:C8:14:73:77
         inet addr:192.168.1.1  Bcast:192.168.0.255 \ Mask:255.255.255.0
         UP BROADCAST RUNNING MULTICAST  MTU:1500 \ Metric:1
         RX packets:1849 errors:0 dropped:0 \ overruns:0 frame:0
         TX packets:1359 errors:0 dropped:0 \ overruns:0 carrier:0
```

```
          collisions:0 txqueuelen:100
          Interrupt:5 Base address:0x340

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:3924  Metric:1
          RX packets:1984 errors:0 dropped:0 \ overruns:0 frame:0
          TX packets:1984 errors:0 dropped:0 \ overruns:0 carrier:0
          collisions:0 txqueuelen:0
```

In this example, `eth0` is the only Ethernet interface, and it has no aliases associated with it. (The loopback interface, `lo`, is not of interest in this discussion.)

To create an alias, issue the `ifconfig` command with arguments describing which interface to alias and the new address:

```
# ifconfig device:aliasnumber address
```

device

The name of the interface to alias is `device`.

aliasnumber

Assign an `aliasnumber` as each new IP address is created.

address

The new IP address is defined when `address` is entered.

Here is an example of creating a new alias:

```
# ifconfig eth0:0 192.168.1.3
```

Along with displaying the current configuration with statistics and creating aliases with new IP addresses, the `ifconfig` command has a number of options, including the ability to delete the aliases the user has created. The user can view a detailed explanation on the man pages. For more detailed information, see the kernel source `Documentation/networking/alias.txt` and the `IP-Alias mini-HOWTO`.

Each IP address could belong to a different instance of the Web server daemon, or the Web server could send requests to a different virtual host based on the IP address.

IP-based hosting does not require a full directory tree for each host, but the user will want to have directories that contain content, logs, and other information specific to each host. Creating these directories in an orderly fashion with names that correspond to their relevant hosts will save time later, when performing tasks specific to each site.

With the examples that follow, we create a `hosts/` directory within the main Apache tree and branch off from there. At a bare minimum, make a single directory for `DocumentRoot`. The user might want to create additional directories that contain logs for errors and user accesses.

As with name-based hosts, add a section within `httpd.conf`, which contains all of the settings that control the behavior of each host. Both types of virtual hosting can use most directives available for a stand-alone server, but a minimal configuration requires only a few.

The following section has been added near the bottom of the existing `httpd.conf` file and defines two IP-based hosts:

```
<VirtualHost 192.168.0.3 >
 DocumentRoot /www/hosts/ipbased1/htdocs
 ServerName www.iptest1.com
</VirtualHost>
<VirtualHost 192.168.0.4 >
 DocumentRoot /www/hosts/ipbased2/htdocs
 ServerName www.iptest2.com
</VirtualHost>
```

A set of basic hosts running with the IP method requires just a few parameters for each host. Here is a breakdown of the directives that have been used with the first host in the example configuration.

<VirtualHost 192.168.0.3 >

Each host requires a section within `http.conf` that contains the configurations specific to that host. The `VirtualHost` statement is followed by the IP address that has been aliased for the host. The host will be accessible at this IP address.

DocumentRoot /www/hosts/ipbased1/htdocs

Within each section, a `DocumentRoot` needs to be assigned that tells Apache where the base directory for a host's `html` content resides. In these examples we have branched off from the `/www` directory and have created a directory for each host. The user can place them anywhere, as long as they are defined in each `DocumentRoot`.

ServerName www.iptest.com

If a host is to have its own domain name associated with its IP address, it needs to be defined here. We recommend having both an IP address and a DNS name for each host; without them, unnecessary lookups and reverse lookups may be done for each connection.

</VirtualHost>

Finally close each section with the virtual host closing statement. This statement is the same for each host. Any directive is permissible in a `VirtualHost` section except those that affect low-level server operations.

Examples of such prohibited directives include the following:

```
ServerType
User
Group
StartServers
MaxSpareServers
MinSpareServers
MaxRequestsPerChild
BindAddress
PidFile
TypesConfig
ServerRoot
```

Besides the standard directives, there are two special directives specific to the `VirtualHost` section: `ServerAlias` and `ServerPath`.

Also, be sure to include configuration for separate log files for each virtual server.

Name-Based Virtual Hosting

Another way to do virtual hosting does not require multiple IPs but uses the `NameVirtualHost` directive. Apache listens on one IP address and serves the Web site based on the Host Header sent by the browser.

The `ServerName` and `ServerAlias` directives are critical to identifying the server that the client requests and are necessary for this method.

Name-based virtual hosting requires the Web browser to send a `Host` field in the HTTP request header (part of HTTP/1.1, supported by version 2.0 browsers and later). Browsers older than 2.0 should be long retired.

IP addresses are a scarce resource; using the name-based method allows multiple Web sites to share a single IP address assignment. In the near future, many sites will be required to renumber their networks to make more efficient use of limited address space. When that time comes, having fewer IP addresses to reassign will be much appreciated. We will examine the IP-based virtual hosting mechanism primarily for historical purposes.

One thing to be aware of with name-based virtual hosts is that if the client does not send the host header, it will get the first virtual host for that IP address. There is a workaround available for supporting older browsers that does not send the host header that uses the `ServerPath` directive. See the Apache Virtual Host documentation for more information. This should be available from the source installation at http://localhost/manual/hosts/name-based.html.

Example of Virtual Hosting with One IP Address

The following is a code excerpt from the `httpd.conf` file used to set up virtual hosting:

```
User webuser
Group webgroup
NameVirtualHost 12.34.56.7
Include /etc/httpd/conf/firstwebsite.comf

#firstwebsite.conf file in /etc/httpd/conf
<VirtualHost 12.34.56.7>
ServerName www.firstwebsite.com
ServerAdmin webmaster@firstwebsite.com
DocumentRoot /home/httpd/firstwebsite/
ErrorLog /var/log/httpd/firstwebsite-error_log
CustomLog /var/log/httpd/firstwebsite-access_log common
</VirtualHost>

#secondwebsite.conf file in /etc/httpd/conf
<VirtualHost 12.34.56.7>
ServerName www.secondwebsite.com
```

```
ServerAdmin webmaster@secondwebsite.com
DocumentRoot /home/httpd/secondwebsite/
ErrorLog /var/log/httpd/secondwebsite-error_log
CustomLog /var/log/httpd/secondwebsite-access_log common
</VirtualHost>
```

NOTE **The IP addresses of the `VirtualHosts` need to match that in the `NameVirtualHost` directive.**

The user can also mix these methods and use port-based virtual hosting all on the same server. Running multiple instances of the Web server daemon requires each instance to have its own unique IP address.

Examples of Virtual Hosting Using Mixed Methods

The IP addresses and comments for each of the `conf` files are as follows:

```
User nobody
Group nobody
Listen 80
Listen 8080
NameVirtualHost 12.34.56.7
Include /etc/httpd/conf/firstwebsite.conf
Include /etc/httpd/conf/secondwebsite.conf
Include /etc/httpd/conf/thirdwebsite.conf
Include /etc/httpd/conf/fourthwebsite.conf
#firstwebsite.conf file in /etc/httpd/conf
#Since the IP matches that in the NameVirtualHost, this
# will use the name-based method
<VirtualHost 12.34.56.7>
ServerName www.firstwebsite.com
ServerAdmin webmaster@firstwebsite.com
DocumentRoot /home/httpd/firstwebsite/
ErrorLog /var/log/httpd/firstwebsite-error_log
CustomLog /var/log/httpd/firstwebsite-access_log common
</VirtualHost>
# secondwebsite.conf file in /etc/httpd/conf
# Since the IP matches that in the NameVirtualHost, this
# will use the name-based method
<VirtualHost 12.34.56.7>
ServerName www.secondwebsite.com
ServerAdmin webmaster@secondwebsite.com
DocumentRoot /home/httpd/secondwebsite/
ErrorLog /var/log/httpd/secondwebsite-error_log
CustomLog /var/log/httpd/secondwebsite-access_log common
</VirtualHost>
```

```
# thirdwebsite.conf file in /etc/httpd/conf
# Since this does not match the NameVirtualHost IP it will
# use the IP based method.
<VirtualHost 12.34.56.78>
ServerName www.thirdwebsite.com
ServerAdmin webmaster@thirdwebsite.com
DocumentRoot /home/httpd/thirdwebsite/
ErrorLog /var/log/httpd/thirdwebsite-error_log
CustomLog /var/log/httpd/thirdwebsite-access_log common
</VirtualHost>
# fourthwebsite.conf file in /etc/httpd/conf
# Since this does not match the NameVirtualHost IP it will
# use the IP based method.  Furthermore, this server will only serve
# the requests that come in on port 8080.
<VirtualHost 12.34.56.78:8080>
ServerName www.fourthwebsite.com
ServerAdmin webmaster@fourthwebsite.com
DocumentRoot /home/httpd/fourthwebsite/
ErrorLog /var/log/httpd/fourthwebsite-error_log
CustomLog /var/log/httpd/fourthwebsite-access_log common
</VirtualHost>
```

NOTE **All the virtual servers are listening on both ports except for the fourth one that is explicitly set to listen only to port 8080. Unfortunately, any port other than the default (port 80) needs to be explicitly requested by the browser, so this may have limited usefulness. Having a valid, matching DNS setup is obviously important, too, but that can be modified by /etc/hosts file to test the setup without modifying the DNS servers.**

Shells and Commands

Sometimes Apache will employ outside programs or modules when confronted with data over a network. These programs can configure incoming requests and data into formats that Apache understands and can process efficiently. Several of these programs are presented in this section.

URL Rewriting mod_rewrite

Apache comes with a rule-based rewriting engine that allows changing URLs after receiving the request; this engine is `mod_rewrite`. As with most powerful tools, a complete understanding is required before using it.

`mod_rewrite` analyzes the URLs it changes in a non-intuitive way. Studying the API will help users understand Apache's approach and how

to develop for it correctly. A description of the API is on Apache's reference for `mod_rewrite`.

The Apache `mod_rewrite` module is very complex, but once a user understands it, almost any URL can be transformed into another one. The module uses a rule-based rewriting engine to rewrite requested URLs. This feature may be found in Apache 1.2 and later. With its powerful URL manipulation mechanism, the Web server can examine every URL that arrives to see if it matches any patterns specified by the rewrite rules. If it finds a match, Apache internally rewrites the URL using the matching rule. Here, the module operates on full URLs in both a per-server context (`httpd.conf`) and a per-directory context (`.htaccess`). To use `mod_rewrite`, add the following line to the configuration file before compiling the server:

```
AddModule modules/standard/mod_rewrite.o
```

Some examples of `mod_rewrite` in action follow.

With a proxy, users can make sure it is used only by those desired. A `403 Forbidden` page can be shown to unwanted users. For instance, users may want to use the proxy server only for URLs with a host name ending in `.anu.edu.au`:

```
RewriteRule !^proxy:http://[^/]*\.anu\.edu\.au/ -  \ [forbidden]
```

Because `mod_rewrite` has access to many server variables, it can be used instead of a CGI or JavaScript to redirect Web browsers based on their type and version. The following example was adapted from Apache's URL Rewriting Guide (the link is provided at the end of this section):

```
RewriteCond %{HTTP_USER_AGENT}     ^Mozilla/3.*
RewriteRule ^/index\.html$         /index.NS.html        [L]
RewriteCond %{HTTP_USER_AGENT}     ^Lynx/.*              [OR]
RewriteCond %{HTTP_USER_AGENT}     ^Mozilla/[12].*
RewriteRule ^/index\.html$         /index.20.html        [L]
RewriteRule ^/index\.html$         /index.32.html        [L]
```

NOTE **The use of `RewriteCond` is one of the most important directives of `mod_rewrite`.**

Following is the syntax:

```
RewriteCond TestString CondPattern [flag]
```

CondPattern TestString

`CondPattern` is an extended regular expression that has some additions.

<, >, and =	When combined with `CondPattern`, treat it as a plain string and compare it lexically to `TestString`.
<	Returns true if `CondPattern` is lexically lower than `TestString`.
>	Returns true if `CondPattern` is lexically greater than `TestString`.
=	Returns true if `CondPattern` is lexically equal to `TestString`.

Put the operator at the beginning of `CondPattern`, like so:

```
<CondPattern
```

Table 3.1 shows some of the options available when using `CondPattern`.

Table 3.1 Options for CondPattern

OPTION	DESCRIPTION
`-d`	Treats TestString as a path name; it tests that TestString exists and that it is a directory.
`-f`	Treats TestString as a path name; it tests that it exists and that it is a regular file.
`-s`	Treats TestString as a path name; it makes sure TestString exists and that its size is greater than zero.
`-l`	Treats TestString as a path name; it tests that it exists and is a symbolic link.
`-F`	Does not deal with path names but checks if TestString is a valid file. Checks to see if the file is accessible via all the server's access controls for that path. Uses an internal subrequest for this check; server performance degrades accordingly.
`-U`	Makes sure TestString is a valid URL and is accessible via the server's access controls for that path. Uses an internal subrequest; server performance lags the more it is used.

To negate the meaning for any of these tests, prefix them with an exclamation mark (`!`).

`CondPattern` has two possible sets of flags: `nocase | NC` and `ornext | OR`. The first, `nocase | NC`, makes the test case-insensitive. `TestString`, teststring, and tEStsTrinG all become the same. `ornext | OR` combines rule conditions, which allows a user to not rewrite the conditions.

Without `OR`, the following code:

```
RewriteCond %{HTTP_USER_AGENT}          ^Lynx/.*                [OR]
RewriteCond %{HTTP_USER_AGENT}          ^Mozilla/[12].*
RewriteRule ^/index\.html$              /index.20.html          [L]
```

would become:

```
RewriteCond %{HTTP_USER_AGENT}          ^Lynx/.*
RewriteRule ^/index\.html$              /index.20.html          [L]
RewriteCond %{HTTP_USER_AGENT}          ^Mozilla/[12].*
RewriteRule ^/index\.html$              /index.20.html          [L]
```

Other Server Variables

Following is a list of other server variables that can be used with `mod_rewrite`.

HTTP HEADERS

HTTP_USER_AGENT

HTTP_REFERER

HTTP_COOKIE

HTTP_FORWARDED

HTTP_HOST

HTTP_PROXY_CONNECTION

HTTP_ACCEPT

CONNECTION AND REQUEST

REMOTE_ADDR

REMOTE_HOST

REMOTE_USER

REMOTE_IDENT

REQUEST_METHOD
SCRIPT_FILENAME
PATH_INFO
QUERY_STRING
AUTH_TYPE

SERVER INTERNALS

DOCUMENT_ROOT
SERVER_ADMIN
SERVER_NAME
SERVER_ADDR
SERVER_PORT
SERVER_PROTOCOL
SERVER_SOFTWARE

SYSTEM VARIABLES

TIME_YEAR
TIME_MON
TIME_DAY
TIME_HOUR
TIME_MIN
TIME_SEC
TIME_WDAY
TIME

SPECIAL VARIABLES

API_VERSION
THE_REQUEST
REQUEST_URI
REQUEST_FILENAME
IS_SUBREQ

To access any environmental variable, use the following syntax:

```
%{ENV:variable}
```

where `variable` is the environmental variable.

Apache looks up the variable through internal structures. If it does not find it there, it uses `getenv()` from the Apache server process.

%{HTTP:header}

Use `%{HTTP:header}` where `header` is any HTTP MIME-header name to find the value of that header.

%{LA-U:variable}

If you want to determine the value of a variable that is actually set later in an API phase, use `%{LA-U:variable}`. It performs an internal (URL-based) subrequest and determines the final value of the variable.

%{LA-F:variable}

A synonym of `%{LA-U:variable}` is `%{LA-F:variable}`. Rather than performing a URL-based subrequest, it performs an internal, file name-based subrequest.

RewriteRule

`RewriteRule` is the directive that completes the tasks in the preceding example, and it is used often with `mod_rewrite`.

The syntax is as follows:

```
RewriteRule Pattern Substitution [flag]
```

where `Pattern` is a POSIX regular expression in Apache versions 1.2.x and later. The `RewriteRule` can be applied more than once. `Pattern` applies to the URL as it stands when it reaches `Rewrite Rule`, not the URL originally requested. So if there are problems getting `RewriteRule` to work properly, look at what has been done to the URL already to see if it has been changed to something unrecognizable. As with `RewriteCond`, `(!)` can be used as a `Pattern` prefix.

Substitution

Substitution is the string that replaces the URL pattern matched. Table 3.2 gives a list of substitution flags and their respective actions.

`RewriteRule` uses the original URL and URI in two special variables called `SCRIPT_URL` and `SCRIPT_URI`.

Table 3.2 Substitution Flags

FLAG	ACTION
`Redirect\|R [=code ]`	Prefixes Substitution with http://currenthost[:port]/, which forces an external redirection. If code blank is left, the HTTP response 302 (MOVED TEMPORARILY) is given. "code" can be any number in the range of 300 to 400.
`Forbidden\|F`	Immediately sends back an HTTP response of 403 (FORBIDDEN).
`gone\|G`	Immediately sends back an HTTP response of 410 (GONE).
`proxy\|P`	Forces Substitution to become a proxy request and redirects it through the proxy module. All rewriting rule processing stops here. If you do not have the proxy module compiled, this flag will fail.
`last\|L`	Stops the rewriting process at the flag.
`next\|N`	Reruns the rewriting process from the first rewriting rule. It does not use the original URL, however, but the URL as it stands after processing. An infinite loop can be created with this flag, so be careful.
`chain\|C`	Groups the current rule with the next rule. The effect becomes apparent if a rule does not match because rewriting skips all following chained rules.
`type\|T=MIME-type`	Changes the MIME-type of the target file to MIME-type.
`nosubreq\|NS`	Forces the rewriting engine to skip a rewriting rule if the current request is an internal subrequest.
`nocase\|NC`	Makes pattern case-insensitive when matching pattern against the current URL.
`qsappend\|QSA`	Instead of replacing the current string, it appends a query string part in Substitution to the existing string.
`passthrough\|PT`	Looks at the internal request_rec structure and sets the URI field to the value of file name.
`skip\|S=num`	Skips the next num rules when the current rule matches.
`env\|E=VAR:VAL`	Gets an environmental variable named VAR that has the value of VAL, which can contain regexp backreferences $N and %N. Use this flag as often as you wish to set variables.

Any server on a network faces security risks. The next chapter will discuss methods of tightening security on your Apache server.

CHAPTER

4

Security

Objectives

- List Apache's authentication methods.
- List common Web server security issues.
- List common programs used to attack Web servers.
- Define Apache as a proxy server.
- Describe the role of a firewall.
- Describe basic configuration options regarding security.
- Describe how Apache incorporates SSL.
- Describe Apache's user-level security.

Theory of Operation

Because Apache is a program designed to be connected and to be used over a network, an administrator must be prepared for many security hazards. This section will introduce a number of normal security concerns, common

policies, and a general introduction to steps that can, when implemented, keep an Apache server safe from being hacked.

Security Concerns

You should be very sensible when using cryptography software because simply running a Secure Socket Layer (SSL) server *does not* guarantee that your system is secure!

SSL itself may not be secure. Some think it is, but do you? Here are some situations to consider:

- Have the authors of the various components put in back doors?
- Does the code take appropriate measures to keep private keys private?
- To what extent is your cooperation in this process required?
- Is your system physically secure?
- Is your system appropriately secured from intrusion over the network?
- Whom do you trust?
- Do you understand the trust relationship involved in SSL certificates?
- Do your system administrators?
- Are your keys generated carefully enough to avoid reverse engineering of the private keys?
- How do you obtain certificates, keys, and the like securely?
- Can you trust your users to safeguard their private keys?
- Can you trust your browser to safeguard its generated private key?

If you cannot answer these questions to your personal satisfaction, then you usually have a problem. Even if you can, you may still *not* be secure. Do not blame the authors if the security fails. SSL should be used at your own risk.

Security Policies

System attacks are possible on any network system. Absolute security is impossible, so policies specific to handling these issues as they happen must be in place. A security breach within a large company could affect thousands of users. How quickly the potential chaos can be resolved depends greatly on the preparation and coordination of analysts, managers, and others. Aspects of the policy should also have preventive mea-

sures, such as company-wide standards regarding secure configurations and timely methods of conveying security-related information.

hosts.allow and hosts.deny

One measure you take to lock down a system from remote intruders is controlling TCP wrappers. Two files determine who can utilize the services that are under control of TCP wrappers: `/etc/hosts.allow` and `/etc/hosts.deny`. Using two lists gives flexibility in how to structure security. For tight security or for a system that does not need to offer many services to outsiders, a user can deny access to all clients and then follow through with the `hosts.allow` file to permit only a narrow range of clients to get through. An open configuration can allow all clients and then limit certain clients from accessing services. A user has the option of using both methods. Connections that have not been specifically denied are accepted by default.

TCP wrappers have a set of wildcards that allow matching services and clients. One important wildcard needed to get a basic configuration together is `ALL`. It matches everything in the daemon or client fields such that the following:

```
ALL:ALL
```

will match all services and all clients.

A configuration line can contain multiple entries for each field, which must be separated by a space or a comma. What follows is an example with multiple entries for `hosts.allow` that grants Telnet and FTP services to everyone from within the local domain and everyone from `trusted.domain.com`:

```
#/etc/hosts.allow
in.ftpd, in.telnet.d: LOCAL, trusted.domain.com
```

Clients can be denied or allowed in either file as long as it is specified with the `DENY` or `ALLOW` flag at the end of the configuration line. The following demonstrates a denial in the allow file:

```
#/etc/hosts.allow
ALL : untrusted.domain.com DENY
```

What you decide to allow and deny depends on the function of your machine. A dial-up box that does not need to serve any clients might deny everything and then let a few friends through. Systems that serve clients on the Internet or other networks require a more proactive approach because denying everyone is not an option.

motd and issue Files

These files reside in the `/etc/` directory and can be used to alert system users to security issues and to prepare them for changes in the system. The issue file contains the text that appears over the login prompt, and the `motd` file contains text that will be presented to each user after login. Configuring is simple; add the desired message in `/etc/motd` or `/etc/issue`.

U.S. Encryption Export Laws

Encryption is a method to secure data. It is crucial to any type of network security. Unfortunately, encryption is not built into the IP protocol; therefore, it is good to use some type of add-on encryption protocol for any sensitive information that travels over a network, especially the Internet. Protocols to consider are PGP, SSL, and IPsec.

At the time of this writing, the regulations pertaining to encryption are undergoing change. Currently, any use of strong encryption (key lengths greater than 40 bits for the RC4 method and 512 bits for RSA) is limited in regard to how it is used or distributed to places outside the United States. There is much controversy over this law, and some changes are occurring that may allow stronger exportable encryption in the future. Until then, the safest approach is to use weak encryption for any aspect of an implementation that could cross U.S. borders.

Authentication

Apache has, by default, multiple authentication methods available. There is a handy password authentication method, an access file authentication method, an anonymous access method, and an as-yet unused digest authentication method. The digest authorization method implements the MD5 specification of digest authentication and is not currently supported by many browsers. For more information on digest authentication, please read the `mod_digest` module information in the Apache Reference Manual that generally comes with the installation. This is usually located in the `/usr/local/apache/htdocs/ manual/mod/mod_digest.html` file.

The methods of authentication discussed in this chapter are as follows:

- Password authentication
- .htaccess file authentication
- Anonymous access

Unless combined with Apache+SSL or Apache using `mod_ssl`, all the passwords sent to the server are in clear text. This is why we recommend using only the previously listed types of authentication on trusted networks or when combined with another type of encryption scheme.

Securing Apache

A Web server in normal use is exposed to the world. Millions of people visit Apache-hosted sites on a given day, and of all those people, a certain percentage will want to do more than just view the content provided by the server.

While no system connected to a network can be totally secure, the combination of Apache and Linux is quite distinguished in terms of security. The fact that Apache has a good security record may be surprising, given that it is freely available. Still, even some expensive commercial Web servers cannot come close to Apache's security record. Of course, server management will make a big difference in how easy it is to defeat security measures.

Security is not just about configuration and hardware. When running any system, especially one exposed to the Internet, you must diligently keep yourself up to date on the latest security exploits and take precautionary measures. One of the best ways to help maintain a secure system is to be proactive, read product updates and security alerts, perform security audits on your system, and make sure that employees follow their security protocols.

This section focuses on guidelines that will help keep Apache secure and points out some issues, such as CGI scripts, that could lead to problems if not watched closely.

Apache User

It is not the best practice to run the Web server as `nobody`, the installation default, because it is important that Apache is protected from other tasks running as `nobody`.

The recommended approach is to create a new, relatively unprivileged user, say `www-data`, and run Apache as that user.

Many people think `nobody` means no user, but that is not true: It is one specific user named `nobody`. Some people make files owned by `nobody`, assuming that no user is allowed to access them. This is an inaccurate idea, and this practice is not recommended.

Vulnerabilities

The first thing to do when securing a system is to assess its vulnerabilities. Things to check include password policies, hostile programs, spoofing, and buffer overruns.

Passwords

An ideal password would contain at least eight characters and be a random mix of letters and numbers, with the characters being both uppercase and lowercase. There are not many efficient ways to crack a good password. Unfortunately, the best passwords are also the hardest to remember. Most people have more than one password that they use on a regular basis, and they choose things they will not forget. Common passwords are typically the name of a spouse or relative or other words to which a person is frequently exposed. The trouble with passwords like these is their vulnerability to dictionary attacks or even to someone trying educated guesses at the password prompt.

For the `/etc/passwd` file to be world-readable is common because many non-privileged programs have relied on being able to read that file. As a result, someone can grab the file and do a brute-force attack on the encoded passwords.

The passwords people tend to choose, though, amount to only a small percentage of the possible combinations. This allows a malicious system hacker to simply use a dictionary instead of a random character generator, and you might be surprised by the success rate.

The Shadow Suite

The Shadow Suite is a replacement for utilities used to create and maintain security settings about the users of the system. The Shadow Suite addresses several vulnerabilities that existed in previous password utili-

ties, such as the /etc/passwd file being world-readable. The Shadow Suite also provides enhanced functionality for password management.

Some of the Shadow Suite's features include the following:

- Encoded passwords are accessible only by root.
- Account information can be aged, meaning that users are automatically prompted to change passwords from time to time, or temporary accounts can be created.
- Users are required to create good passwords.
- Utilities for account/password management have been improved.
- A configuration file to set login defaults (/etc/login.defs) is included.

The format for entries in /etc/shadow follows:

```
<user name>:<password>:<last change>:<allow changes>:<require
changes>:<warning>:<expiry:days> <disabled:reserved>
```

User name	The username that corresponds to the username in /etc/passwd.
Password	The account's encrypted or encoded password.
Last Change	Starting from January 1, 1970, the number of days since the password was last changed.
Allow Changes	The number of days before the user can change the password. A setting of -1 allows the user to change the password at any time.
Require Change	The number of days before the user must change the password. By setting the value to a large number, such as 60,000, you can essentially disable this feature.
Warning	The system's indication of a number of days during which the user needs to update the password before it expires.
Expiry	If the user has not changed his or her password, the number of days after the <require change> date has passed until the account is disabled.
Disabled	The number of days the account has been disabled since January 1, 1970.
Reserved	Reserved for use by the Shadow password software.

Hostile Programs

Hostile programs are programs that can do harm to your system. They include Trojan horses, viruses, and worms.

Trojan Horses

Trojan horses are named after the method the Greeks employed to circumvent the security of Troy. According to myth, after a futile 10-year siege, the Greeks built a large wooden horse and presented it to the Trojans as a peace offering. The Trojans accepted the horse and brought it into their heretofore impenetrable city. The Trojans did not know that the wooden horse had a belly full of Greek soldiers who, in the dead of night, killed the city guards and opened Troy to the rest of the invading force.

Like the story, a Trojan horse program looks harmless and claims to do something useful or entertaining for the user. When the program runs, it appears to do this certain task, but its primary, hidden function is to perform another, possibly malicious, task.

A widely used Trojan horse replaces the login program. The Trojan horse program appears to perform the login command. In reality, the program captures the password of the user that logged into the system, mails the password to another account, logs the user in (or gives an error message and surrenders to the real login program), and removes itself from the system. This type of Trojan horse is typically installed by an intruder, and its presence is not immediately known to the user.

Another example might display an amusing animation or offer to play a game. While the user is being entertained, the program goes about its task of exploring the system, sending contact information to its master, or vandalizing the host system.

Viruses

Viruses are another form of attack. A virus is a piece of code that might enter a system as part of a program. Once in the system, it may hide itself in memory and attach itself to every program that runs or every program file it finds on disk. Some viruses are deliberately destructive in nature, destroying the master boot record on your hard drive, erasing programs, or simply rendering programs inoperable. Other viruses may not intend to cause damage but merely annoy by, for example, displaying a message or animation on your monitor. Frequently, due to poor coding technique, this class of virus can consume excessive resources and inadvertently cause downtime or data loss, sometimes because of simple user panic.

Worms

Worms are yet another form of attack. Unlike a virus, a worm enters a computer system as a stand-alone program not attached to other pieces of code. Once in the system, the worm goes about its business. Like a virus, a worm may be intentionally destructive, or it may be benign. Sometimes a benign worm, like a benign virus, will go out of control and cause unintentional damage or resource depletion.

Spoofing

There are different types of spoofing attacks that prompt software into an inappropriate action by presenting misleading information to that software. In TCP spoofing, Internet packets are sent with forged return addresses, and in DNS spoofing, the attacker forges information about machine names and network addresses. In addition to these two spoofing attacks, there is Web spoofing, or the "man in the middle attack." It allows an attacker to "shadow copy" the World Wide Web, which lets the attacker monitor all the victim's activities, including account numbers or passwords. While this attack mainly affects end users, you should still know what it is and how it works to protect your Web site from being spoofed.

Buffer Overruns

Buffer overruns are the source of most attacks to Linux systems. Their vulnerability comes from certain coding errors that are difficult for programmers to avoid, causing many programs to contain these errors. A buffer overrun occurs when certain variables receive more data than the programmer anticipated. If that variable happens to be responsible for storing input that someone can modify, a skilled attacker can induce the error condition at will, causing the program to crash or to function in ways that weren't part of its original design. When an important and/or high-availability program contains one of these errors, it can be used as a path of attack.

Attackers seek certain types of programs when looking for buffer-overrun candidates. Programs that have `setuid` privileges are favored by attackers. Setuid programs have root-level access, but although they use the privilege only to perform small tasks, they have the ability to do almost anything nonetheless. When an attacker induces a buffer overrun within a setuid program, the program can crash in a way that leaves a root shell for the attacker. Another possibility is acceptance of errant behavior finessed from the program, as if it originated from root.

Buffer overflow problems have recently been exploited in such popular network daemons as `qpopper`, `named`, `wu-ftpd`, and many others. Candidates for the buffer overrun exploit can include older versions of ProFTPD and `wu-ftp`. Due to insufficient bounds checking in these programs, it is possible to subvert an FTP server by corrupting its internal stack space. By supplying carefully designed commands to the FTP server, intruders could force the server to execute arbitrary commands using root privilege. Thus, intruders who are able to exploit this vulnerability can ultimately gain interactive access with root privilege to the remote FTP server. The most vulnerable systems were the ones with the ftpd software installed and enabled by default.

One temporary solution to this attack is to disable any world-writable directories to which the user may have access by making them read-only. This action will prevent an attacker from building an unusually large path, which is required to execute these particular attacks. The other preferred solution is to upgrade the programs with patches that address the potential buffer overruns.

Of even more interest, CERT reported a buffer overrun in early 1998 in `<cfg_getline()>` that possibly allowed malicious users to gain access, not as root, but as the user of Apache. This is all the more reason to run Apache as any non-privileged user, other than `nobody`.

Those using pre-1.2.5 versions of Apache are susceptible to this buffer overflow in `<cfg_getline()>`. `<cfg_getline()>` is a function that the Apache core and several Apache modules use to read certain types of files from disk. Some examples of the type of files that read with this are `htaccess`, `htpasswd`, and `mod_imap files`.

It is possible to create a sequence of data such that a buffer overflow occurs while `<cfg_getline()>` is reading from a file. If someone has access to create any of these types of files on the server, that hole is generally exploitable to gain full access to the user that Apache runs as. On most systems, this is of little consequence because many users already have such access. If, however, the server is secured so that the user has no access to the server other than to create and modify files (for example, an "FTP only" account with no ability to create CGI scripts), this could allow increased access to the server.

Security Issues with CGI

Because CGI scripts are executables, they are subject to the same security vulnerabilities as normal programs. What makes CGI scripts especially

dangerous is that anyone in the world with a Web browser and Internet connection can execute programs on any public Web server.

The first step in securing CGIs is to secure the server; Apache should be run as a non-trusted user with secure permissions on key files. The real key to keeping CGI scripts safe is to eliminate vulnerabilities in them.

System Calls

Most CGI programs perform system calls, such as reading from or writing to a disk file, or perhaps executing other programs. If any of the variables passed to the CGI are used in formulating the parameters to these system calls, malicious users can manipulate them to do undesirable things.

Certain commands are very dangerous; the `eval` command is one of them. `eval` lets a script execute an arbitrary command given in a variable. A CGI script could take the name of the command to run as a parameter, execute it, and show the output. This could give a system cracker all the ammunition necessary to break into a system.

Buffer Overruns

C programs suffer from vulnerability to buffer overruns. When C programs read data, they must allocate enough room in memory to hold it. When more data is read in than room is available, the excess data can be executed as code. If writing scripts in C, ensure they are free from buffer overruns by checking the amount of data that comes into the program. Although this is relatively difficult to exploit, it is worth watching.

The Apache Proxy Server

There is much confusion over the role of a proxy on a network. Proxies are often associated with a firewall and network security policy. The proxy is not usually the firewall itself, but it is often used with a firewall. In some implementations, such as SOCKS, though, it can be used as the firewall itself. The main purposes of proxies are to enhance performance of Internet requests by caching files commonly accessed and to play a role in allowing/denying access to other network resources on either side of a firewall. The proxy will often sit on the firewall itself. Much like a firewall, the proxy will have access to the outside. When the user configures a client to use a proxy, it may seem as if there is a direct connection with the outside network. With a proxy this is not really the case. When a request is

made to the outside network, the client is actually giving the request to the proxy. If the proxy has the information requested, it serves it back to the client. If not, the proxy makes the formal request to the specified server and receives the information. At this point, the proxy provides the data to the requesting client. Even though the original request to the proxy is via HTTP, the proxy may use any protocol to actually retrieve the requested document(s).

An HTTP proxy cache accepts requests for objects that people want to download and handles their requests. If the user wants to download a Web page, he or she asks the proxy server to retrieve the page. The proxy then connects to the remote server and requests the page. It then transparently streams the data through itself to the client machine, but at the same time it keeps a copy. The next time someone wants that page, the proxy simply reads it off disk, transferring the data to the client machine almost immediately.

The following are some common uses for a Web proxy:

- Permit and restrict client access to an Internet based on the client IP address
- Cache documents to increase internal network efficiency
- Control access to the network based on the requested URL
- Provide network access for networks using firewalls for separate networks
- Provide a gateway for services other than HTTP and networks without DNS

Figure 4.1 depicts a proxy-firewall combination, which can be used to provide network access.

Advantages of the Apache Proxy

Why should a user employ the Apache proxy server? First, the Apache proxy server is free. Many Web sites across the Internet already use the Apache Web server. Web site hosting companies and individuals using the Apache Web server are familiar with the software and have adapted its abilities to their security needs. Using the Apache software as a proxy server requires almost no learning curve.

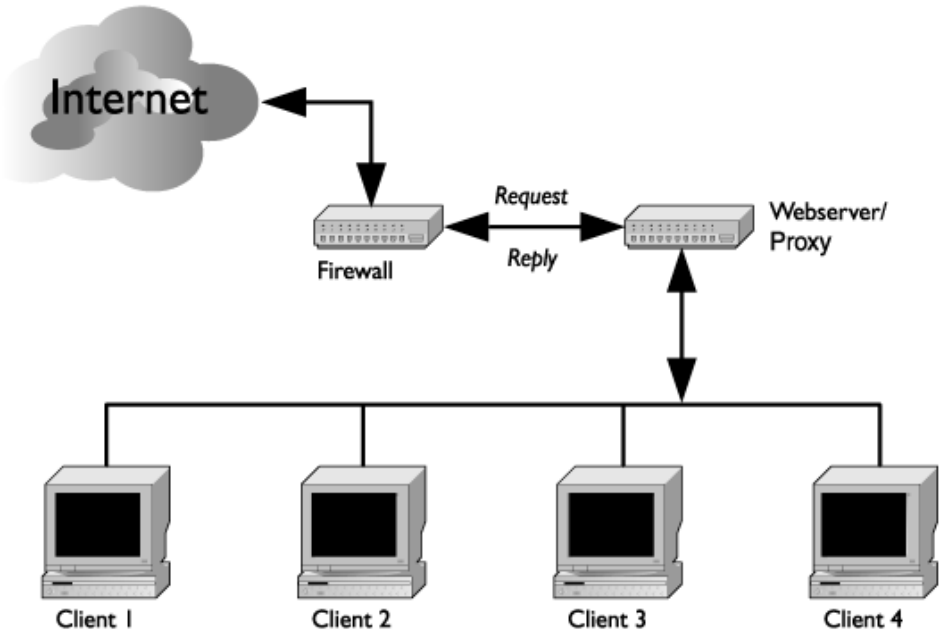


Figure 4.1 Proxy and firewall combination.

Apache can be used for an `http` caching proxy by using one of the modules that comes with the Apache source distribution: `mod_proxy`; this is a plus when the Apache proxy is compared to other proxy servers. Many proxy servers require tedious configurations just to be set to maintain network security. Some even require many hours of work to attain an adaptive network usage.

Obtaining the Apache Proxy

Apache is freely available via download from www.apache.org/, which provides links to Apache mirrors in more than 10 countries. If the user is seeking information on Apache and XML, then www.apache.org will be beneficial. Direct links to full Web pages detail how, why, and what if. The following links are provided:

- Apache Server
- XML-Apache
- Java-Apache

- mod_perl
- PHP
- Apache Tcl

Obtaining Documentation

Apache documentation is accessible from http://httpd.apache.org/. Mirrors can also be accessed from this Web site. The Apache decantation group excelled at covering possible problems and configuration HOWTOS. The following and many more topics are covered in the Apache documentation:

- Main configuration files
- Syntax of the configuration files
- Loading modules
- Directives (capabilities)
- Access
- Log files
- Starting Apache
- Errors during startup
- Starting at boot-time

mod_proxy

Generally Apache uses the `mod_proxy` module to act as an intermediary between the client and the Web server that actually contains content. As a proxy, Apache will decide if it should save time and bandwidth and use a cached file to fulfill a request or if it should grab the real file from the server and pass it back to the client. As of Apache 1.3, `mod_proxy` is still only an HTTP/1.0 proxy.

Firewalls

Firewalls get their name from the part of a car chassis that separates the passenger compartment from the engine compartment. If there is an engine fire, the firewall helps to protect the passengers. A network firewall separates parts of a network, such as a business network, from the Internet. Instead of protecting from fire, it protects from intruders and allows control of the information that travels between networks. Having a firewall is

generally a good idea because it allows an administrator to centralize many security policies to one system rather than having to worry about every single machine on a network. Presence of a firewall does not guarantee absolute security but can enhance it significantly if the firewall is well maintained.

To make the firewall effective, all traffic entering or leaving a site must pass through it. In some cases, multiple firewalls are necessary when a company has multiple connections, as routing all traffic through a single firewall may not be feasible. In the end, all access points into and out of the internal network will need a firewall to guard them.

In addition to security, the firewall also provides network address translation (NAT). Because Internet network addresses are becoming very scarce, a firewall running NAT enables a company to use a few legal IP addresses for essential Internet systems (DNS, mail, Web, FTP, etc.) while the private network uses the IP address space reserved for private use by IANA. With NAT, as data packets pass through the firewall from systems using the private IP addresses, the firewall translates them into registered IP addresses. Conversely, when the reply packets return from a system on the Internet, the firewall performs a reverse translation.

Types of Firewalls

Although firewalls can perform many different functions, we will discuss three particular types:

- Packet filtering
- Proxy serving
- IP masquerading

By combining these three functions, we can create an effective firewall that will help maintain a tight, secure network.

Packet Filtering

By running a firewall as a packet filtering router, the firewall examines each packet of data (a datagram) against a set of predefined rules. The filtering rules are based on packet header information, which is available to the IP forwarding process of the kernel. In the IP packet header, you will find the following information:

- IP source address
- IP destination address
- Protocol: UDP, TCP, ICMP, or IP tunnel
- Source port for UDP or TCP
- Destination port for UDP or TCP
- ICMP message type
- Incoming interface of the packet
- Outgoing interface of the packet

Based on the information that the IP packet headers provide, rules can be defined that either allow or deny packets to reach their destination.

For example, incoming SMTP traffic can be permitted to go only to a designated mail exchanger host on the internal network and deny all incoming telnet. At the same time, incoming HTTP and FTP requests to a specific Web or FTP server can be allowed.

Why Packet Filtering?

Packet filtering is useful not only for allowing, denying, and routing incoming requests, but also because it provides a first line of defense against some types of attacks, most notably spoofing, fragment, and source routing attacks.

Spoofing. In an IP spoofing attack, the malicious person sends packets from outside that are designed to resemble packets that would have originated from an internal host. In other words, the packets are disguised to look as though they originally came from the internal network behind the firewall. To combat spoofing, the firewall simply checks the packets that arrive on the external network interface, and if their source address is that of an internal host, it refuses to admit the packets. Hence, this sort of attack is simple to identify and nullify.

To turn on the kernel's built-in anti-spoofing capabilities, run this command:

```
<# for in /proc/sys/net/ipv4/conf/*/rp_filter; do echo 1> $f; done>
```

Fragment and source attacks. Packet filtering also prevents tiny fragment attacks and source routing attacks. Tiny fragment attacks use the IP fragmentation feature to create very small packets that split the TCP header information into separate packets, or fragments. The success of this

attack relies on fooling your filtering rules into permitting the first fragment to pass and subsequently allowing the rest of the fragments to pass unchecked. To protect against this sort of attack, set the rules to deny all TCP packets that have the IP `FragmentOffset` header set to 1.

To prevent source routing attacks, set the rules to drop any packets that do not contain the source route option.

Proxy Server Firewalls

The role of the proxy server is to monitor inbound and outbound traffic. One benefit of using a proxy server is that it creates detailed logs of all data transfers. Also, as they can examine the payload of a packet, proxy servers can filter inappropriate content.

There are two types of proxy servers:

Application proxy servers. For each protocol that needs to run through the proxy server, the application proxy server provides separate services. One such example of a proxy server is a squid proxy server that provides HTTP and FTP proxies.

SOCKS proxy server. This server provides generalized proxy services for applications that can use a SOCKS proxy server.

IP Masquerading Proxy Servers

In order to use NAT to its full potential, IP masquerading can be used. As a firewall tool, NAT provides the following benefits:

Hidden internal network. Because the internal networks are not routed over the Internet, networks behind the firewall are hidden from the rest of the world (i.e., the Internet). The hidden network cannot be examined externally and so security is greatly enhanced.

Utilization of fewer valid IP addresses. When the firewall uses NAT, the network needs fewer registered IP addresses.

Thus, IP masquerading allows all internal machines to access the Internet from behind the firewall using the firewall's IP address. As a result, the Internet sees only the firewall machine and nothing beyond it. Breaking through a properly secured Linux IP masquerading host and into the internal network is extremely difficult.

Firewalls and Network Architecture

There are different ways to set up networks, and the architecture ultimately depends on the requirements of the network. The simplest network has two components: an internal network and an external network with a firewall between the two.

Still, corporations have needs that exceed the simple two-network design. As a result, separate networks are created to house the public access servers. Often referred to as the demilitarized zone (DMZ), this network contains the mail, DNS, FTP, and Web servers. For security, the DMZ is configured to provide limited access from the Internet to specific public information servers.

As different services and requirements are needed, new portions of the network can be designed. The bottom line for creating all of these separated networks is to allow the application of different sets of security rules for each network.

Securing the Firewall Machine

Obviously, it is of great importance to make sure that the firewall machine is itself locked down and not left open to attacks. The first step in doing this is to turn off all necessary services. Then, rename or delete the current `</etc/inetd.conf>` file and create a new one with this line:

```
telnet stream tcp nowait root /usr/sbin/in.telnetd in.telnetd
```

This allows only for Telnet service for remote management of the firewall system. After creating the new `inetd` file, send the `SIGHUP` signal to `inetd`.

The `/etc/passwd` and `/etc/shadow` should contain an absolute minimum of accounts belonging to the administrator, root, bin, daemon, sync, shutdown, halt, and operator. Furthermore, the system should not be part of an NIS domain, and the machine should be used only for firewall administration and operation.

Superfluous daemons (`nfsd`, `dhcpd`, `named`, `atd`, etc.), as well as X Windows, should not run on the system. To disable particular daemons from starting at run level three, go to `/etc/rc.d/rc3.d` and rename the

links starting with an uppercase S to a lowercase s. To disable the `mysql` daemon, change the link from `S90mysql` to `s90msql`.

In Telnet, direct root login should be prohibited, and the passwords should be secure (minimum lengths, non-dictionary words that are mixed-case alpha-numerics). Change the passwords frequently, such as every three months.

For further protection, consider disabling Telnet, installing SSH instead, and connecting through SSH from designated secure hosts only.

What to Do If Attacked

The following list includes some steps you should take if your system comes under attack:

- Gather as much information as possible.
- Determine what part of the system was vulnerable.
- Make appropriate changes.

If your network has been attacked, you need to gather as much information as possible. Start with the system logs to try to determine how and when the attackers entered. Some attackers attempt to remove evidence of their entry from the system logs, so be observant for the disappearance of information as well. Careful examination of the logs will provide you with the attacker's method of entry (which security loophole the attacker came through). Once you know the vulnerability, you can check the different resources (Bugtraq, CERT, or distribution sites) to see if a patch has been made available for this loophole, and if so, apply it to your system. If you cannot find a patch, you can certainly disseminate the existence of a security loophole, and a patch will usually come out in a short time.

Fortunately, most log entries that relate to security issues on an up-to-date system represent failures on the part of the attacker. The Linux community usually responds very quickly to system weaknesses and provides regular updates that render many intrusion tools useless. Addressing failed attempts usually involves blocking the offending IP and contacting the service provider that owns the originating address. It would also be a good idea to check CERT and Rootshell to see if there are any new exploits for the targeted service. No system can be completely secure, and, from

time to time, an intruder will be successful and perhaps even gain privileged access. In a worst-case scenario, finding an intruder on a system might require dropping network interfaces, rebooting the server, or even doing a major restore of system files.

Handling an Attack That Is in Progress

If you happen to catch an attack while it is in progress, the best thing to do is to remove the system from the network until you can resolve the weakness. You might unplug the Ethernet or hang up the modem; this isn't always an option if the system absolutely needs to be available to other users during the time of the attack. If you cannot bring down the network, you can disable the service from which the attack is originating by commenting it out in `inetd.conf`, followed by a `kill -HUP pid`. If that service is the one that must stay up, try to block a range of the attacker's IP in TCP wrappers. Regardless of how it is done, you must modify your situation quickly to minimize damage.

Notify Service Provider or Local Authorities

If you were able to get the IP address of the attacker (this is not difficult in most situations), you may be able to use the `whois` utility to find out who owns the domain block from which the attack originated. If the owner is not the attacker, you will most likely receive some assistance.

Restore the System

Recovery from attack will be much easier if you are using tools like `tripwire` to point out files that have been compromised. Having good backups of critical system files (`login`, `ls`, `bash`, etc.) is important, as they are often used as back doors or Trojan horses by those who have gained privileged access. If at all possible, keep the system disconnected from the network until you can discover the cause of the security breach.

Block IP Addresses

Once you have determined the domain from which the attack originated, it might be a good idea to block as much of it as possible. If the attacker has a dynamic IP address and you blocked only one IP address, the attacker could return at another time with a slightly different address and regain access. Blocking as much of the attacker's domain as possible will prevent this.

Counterattack?

Never! Launching an attack is a very bad idea, not to mention illegal. Doing so could make it impossible to determine the origin of attack. There

is a very high probability that the origin is simply some random machine on the Internet that has been compromised. An entire chain of compromised systems could be between you and that system. Before you get any ideas of launching a counterattack, remember that it is probably not the correct machine. If you feel that something needs to be done, gather as much information about the attacker's origin and report it to the origin's service provider or to the authorities.

Password Protection

In previous modules you set up password protection for your server. A set of users can be defined, especially for Web access using the `htpasswd` program. This gives the Apache administrator full control over which users can access specific resources on your Web site(s). If you have root access to your machine, you can also use the standard `useradd` command to add users along with the `groupadd` command to add groups. A user can reach an extensive level of complexity in the administration Web permissions by combining users and groups. No matter what username and password a user has, you can require that the user belong to a given group before allowing access using the `Require Group groupname` directive. Each method has its advantages and drawbacks.

The `allow, deny` directives can be combined with password protection to further lock down a server by keeping all but the most qualified users away from coveted resources. `allow, deny` can be combined with `Satisfy` to offer two ways to reach resources, a valid IP address or a valid username/password combination.

Password protection is something you can pass down to people you do not want to access `httpd.conf`, due to `.htaccess` and `.htpasswd` files. In this context you are able to give other people more power, which results in less time spent restarting Apache and fewer requests in your inbox, leaving time to focus on more important security matters. The downside to `.htaccess` files is, of course, increased load on your server. It is a good idea to make judicious use of `.htaccess` files to attain a balance between server performance and customization for your users.

Another way to allow password protection on your server is to give your users access to CGIs that mimic the username/password access Apache offers. PHP, for instance, can send an unauthorized header to the top of a Web page, prompting a username/password window.

Performance Monitoring

Performance monitoring can serve many purposes. While you can view it as a reflection of changes, you can also use it to see if your server is being attacked. How could you use `mod_status` to detect troublemakers?

Base Systems

Many different people can be working on the Web site that is served by Apache. There could be many different groups of people, with each group (and/or person) needing unique permissions. This section will describe the ideas of access control, keeping a structure to security policies for an Apache install, how to use Apache as a proxy, and much more.

Apache, Users, and Groups

If you install Apache from source and do not specify a user, Apache will run as the user `nobody`. Running Apache as `nobody` is dangerous because other processes run as `nobody` already. In the event of a security breach, trying to figure out which `nobody` actually affected your system becomes quite difficult. Remember also that any CGI script that Apache runs does its business as the same user as Apache.

The obvious solution is to run Apache as a different user. We recommend configuring Apache to run as `webuser`, as a member of `webgroup`.

Before Apache can run as `webuser` of `webgroup`, you must add that user and group to Linux. First, make sure you are operating as root, then add your group:

```
# groupadd webgroup
```

While still operating as root, add your user:

```
# useradd webuser -g webgroup
```

`webuser` now exists as a member of `webgroup`. If you were running multiple daemons and were very nervous about security, you could conceivably run each daemon as a different user, all as parts of the `webgroup`.

To make sure Apache runs as the user you want, add the following to your `./configure` command when you install Apache:

```
--server-uid=webuser \
--server-gid=webgroup
```

If you have already compiled Apache and it runs as `nobody`, you can also change its user ID and group ID in `httpd.conf`. Just look for the lines:

```
User nobody
Group nobody
```

and change it to the user and group you like best. Then restart Apache.

The preceding assumes Apache is running as `nobody`.

Once you have changed Apache's identity, it is a good idea to see if it can still read the files in its `DocumentRoot`. Change directories into your document root and view those files.

Don't remember where your `DocumentRoot` is? Within your `httpd.conf` file, use your favorite text editor to search for the string `DocumentRoot`. Follow the absolute path that resides between the `DocumentRoot` tags to find your document root.

Permissions

For the majority of the files it handles, all Apache has to be able to do is read them. It must also be able to view the contents of any directories you may have.

Apache needs special permissions in your CGI bin(s), where it needs permission to execute any CGIs you may have. Further, if those CGIs add, modify, execute, or delete any files, you must change their permissions for `webuser` accordingly.

You may want to change the permissions and ownership of some files to enhance security. You may find that root owns most files in Apache's document root directory. If you would rather have `webuser` own those files, you need to use the `chown` command to change the owner.

Remember, you need permissions to change a file before you can change ownership. Therefore, if a file is owned by root and that is the only user that has write access to the file, then you must log in as root to change the file's permissions. The following:

```
# chown webuser platform.html
```

will change the owner of `platform.html` to `webuser`.

Imagine that you want to change the owner of `top_secret` and all the files that reside in it. You would want to use the `-R` option with `chown` to recursively change permissions on all files in `top_secret` and `top_secret`'s subdirectories.

```
chown -R webuser top_secret
```

Access Control

Controlling access to directories is usually a relatively large concern for an Apache administrator. There are several ways to control access to different areas of the Web-space.

AllowOverride

This is a very important directive and, if at all possible, should always be set to `None`. The `AllowOveride` directive tells the server which access directives can be overridden by the `.htaccess` file. If `AllowOveride` is set to anything other than `None`, the server is forced to read every `.htaccess` file in every directory and subdirectory for every request, whether the `.htaccess` file exists. This uses the stat system call and causes the server to do a context switch, perform a disk access to check for the file, and apply the directives in it before it can serve the request, slowing the server. If users need to set their own access, it may be best to use this directive to specify what can be overridden in the `.htaccess` file. Some policy decisions must be made, based on the site's needs. Remember, it is important to recognize the performance consequences that result from enabling the `.htaccess` method.

Order, Allow, Deny

These three directives control access to specific parts of the server. They can be used in the `<Directory>`, `<Files>`, and `<Location>` sections to control access. It is also possible to control access based on client host name, IP address, or other aspects of the request using environmental variables. Use the `Allow` and `Deny` directives to define access (or lack of access) to specific clients. Use the `Order` directive to set the default access state and configure how the `Allow` and `Deny` directives interact.

Following are two examples of the `Order` directive:

`Order Deny,Allow`. Apache evaluates the `Deny` directives before the `Allow` directives. Access is allowed by default.

`Order Allow,Deny`. Apache evaluates the `Allow` directive before `Deny`. It denies access by default.

Notice the lack of space between the commas. The syntax allows separation of keywords with a comma only, meaning no white space.

Following is an example of the `Order` directive working with the `Deny` and `Allow` directives:

```
<Directory "/my/directory/">
    Order Deny,Allow
    Deny from all
      Allow from specialserver.org
</Directory>
```

In this example, only people who come from `specialserver.org` can access `/my/directory/`. The default is to allow people, but the `Deny from all` negates that. Use the `Satisfy` directive in concert with the `Allow` and `Deny` directives. It is really useful only if access to a directory, location, or file is restricted by both username/password and client host address. When this happens, use the `Satisfy` directive to modify the default behavior (which is to require both). Within the directory or `.htaccess` context, `Satisfy any` means that the client only has to satisfy host restriction or enter a valid username and password.

Although we do not recommend using `.htaccess` for access control, the default configuration of Apache allows for it and sets the access to `Deny from all`. If the `.htaccess` method is used, modify the `AllowOveride` directive accordingly. Remember that the changes will apply, and turn on the stat call for all subdirectories.

To use clear text authentication to access the entire Web-space, follow this process.

Add the following lines to the `DocumentRoot` Directory block.

```
AuthType Basic
AuthName "Restricted Site"
AuthUserFile /etc/httpd/conf/users
Require valid-user
```

NOTE **These additional directives could also be put in the `.htaccess` file, but, as stated, we do not recommend using the `.htaccess` files for performance reasons.**

Now it looks like this:

```
<Directory />
    Options FollowSymLinks
    AllowOverride None
    AuthType Basic
    AuthName "NonFreeZone"
    AuthUserFile /etc/httpd/conf/users
    Require valid-user
</Directory>
```

To display the usage options of the `httpasswd` program, type the following at the root prompt:

```
root@foo $  cd /usr/local/apache
root@foo apache $  bin/htpasswd -?
```

Create a new user and set the password:

```
root@foo apache $  bin/htpasswd /etc/httpd/conf/users test
New password:
Re-type new password:
Adding password for user test
root@foo apache $  bin/apachectl restart
```

The `AuthUserFile` line found in the `httpd.conf` file must correspond to the `httpasswd` file. If the `httpasswd` file does not exist, use the `-c` option for the `httpasswd` command to create the file.

```
root@foo apache $  bin/htpasswd -c /etc/httpd/conf/users test
```

Now We Can Test It

Open up a browser on the client machine, and try logging into the Web site http://domain.suffix. You will be prompted with a `password required` dialog box (see Figure 4.2).

This information could be applied to any directory, but by doing it this way, a valid-user login is required to access the entire site. For more information, see the online documentation for the `Require` directives and those directives associated with it.



Figure 4.2 Password Required dialog box prompt.

Anonymous Access

Anonymous access uses the `anon_auth_module` to allow anonymous users to sign in with a default username and a valid e-mail address in order to access information. Apache checks the e-mail address validity by looking for both the "@" and the "." symbols. The anonymous access method is good for tracking users because it does not rely on cookies, but rather logs the e-mail address entered by the user as the client moves around a site.

When anonymous access is selected, it is necessary to select the valid user IDs using the `Anonymous` directive. For example:

```
Anonymous test welcome joe_user
```

This would allow anybody logging in with `test`, `welcome`, or `joe_user` to access the system.

Setting Up the Apache Proxy

This section provides walk-throughs for setting up Apache as a proxy. The different configurations and security issues that can occur, when using Apache as a proxy, will also be discussed.

Proxy Specific Directives

Table 4.1 shows some of the Apache proxy common directives.

Table 4.1 Apache Proxy Common Directives

DERIVATIVE AND SYNTAX	DESCRIPTION
`ProxyRequests on`	Turns the proxy functionality on.
`ProxyRequests off`	Turns the proxy functionality off.
`ProxyRemote remote-server = protocol: //hostname [:port]`	Specifies a remote proxy for the given protocol. (If the * option can be given as the remote server, the server will be contacted for all requests.)
`ProxyPass path url`	Allows servers behind a firewall to be available via Internet without changing firewall rules or compromising security settings.
`ProxyPassReverse path usr`	Allows the Apache proxy module to share its load with proxy servers.
`ProxyDomain domain_name`	Specifies the default domain of which the proxy is a part. (Only useful for Apache proxy servers within intranets.)
`NoProxy {Domain \| SUBNET \| IP-Address \| HOSTNAME}`	Only useful for Apache proxy servers within intranets; specifies a list of domains, IP-addresses, subnets, and host names to be served directly by the Apache server.
`ProxyBlock {* \| wird \| host \|domain [word \| host \| domain}...`	Specifies a list of users for which HTTP, HTTPS, and FTP requests are blocked. (Option available in Apache 1.2 and later.)
`AllowCONNECTport [port]...`	Overrides the default HTTPS and snews ports. (This option allows connections to the listed ports. Option available only in Apache 1.3.2 and later.)
`ProxyReceiveBufferSize bytes`	Defines an explicit network buffer size for outgoing HTTP and FTP connections. (Must be higher than 512 or set to zero.)
`ProxyVIa {on \| off \| full \| block}`	Used to control the flow of proxy request along a chain of proxy servers.

Table 4.2 shows some Apache proxy caching common directives.

`ProxyPass` is a powerful and very useful directive. If a small company has a network that consists of six computers the only machine accessible to the outside world is the Web server/proxy. The `ProxyPass` directive will allow a client to be mapped so that it can be accessed from an outside world as well. In this scenario the Web server/proxy is a masquerading host. Thus, the network is functioning with one IP address.

Table 4.2 Apache Proxy Caching Common Directives

DERIVATIVE AND SYNTAX	DESCRIPTION
`CacheSize kilobytes`	Is the size of the cache file in kilobytes. The location of this file is determined by the `CacheRoot` directive.
`CacheRoot directory_name`	Sets the caching directory.
`CacheMaxExpire hours`	Defines the `Max` number of hours for which cachable `HTTP` documents will be retained.
`CacheGcInterval number_of_hours`	Checks cache every `X` hours and clears cache if cache is greater than `CacheSize.`
`CacheDirLevels`	Sets the number of subdirectories in the cache.
`CacheForceCompletion`	Specifies the cache percentage at which a canceled request will still be cached.
`NoCache`	Specifies retrieved documents that should not be cached.

Server-Side Configuration

There are certain configuration files and commands that are issued on the Apache server when using Apache as a proxy. This section describes such steps.

Configuring Apache as a Proxy Server

Many install the Apache Web server while installing their particular distribution of Linux. Normally, distribution releases of the Apache software are not compiled with the `mod_proxy` module. Use the following command to list all modules currently compiled into the server:

```
bash$ httpd -l
```

If the `mod_proxy` module is not compiled into the release, then the user will need to obtain the Apache source and compile including the `mod_proxy` module. The previous method is recommended for an older version of Apache. In the newest version of Apache, just uncomment and edit the following lines in the `httpd.conf` configuration file:

```
## httpd.conf -- Apache HTTP server configuration file
LoadModule proxy_module modules/libproxy.so
AddModule mod_proxy.c
```

```
ProxyRequests On
CacheRoot /var/cache/httpd
CacheSize 5
CacheGcInterval 4
CacheMaxExpire 25
CacheLastModifiedFactor 0.2
CacheDefaultExpire 1
NoCache host1.com eagle2.com etc.com
```

NOTE **The preceding configuration options are not in the given order. They are spread throughout the file.**

This method can be used on a standard or normal install. Both of these methods work well if Apache is being used as a proxy server only. To use Apache as a Web server and a proxy server, a different approach will have to be taken.

Configuring Apache as a Proxy/Web Server

More than one instance of the Apache server can run at the same time. A simple way to configure a system to do this is to copy the existing `httpd.conf` to a file called `proxy.conf`. Then edit the new file to run on a new port (i.e., `8080` instead of `port 80`). Limit access to this proxy server by binding to a private IP-address or by some other method. For example, if the IP range was 257.0.*.*, the user would bind the proxy server to a private address by using the command `BindAddress 257.0.0.1` in `proxy.conf`. This would grant access to the user over the IP-range 257.0.0.1 to 257.0.0255. Another option would be to add the following segment to the `config` file:

```
Proxy Access defined
<Directory proxy:*>
order deny,allow
deny from all
allow from 257.0.0.0/255.255.255.0 198.162.0.0/255.255.0.0
</Directory>
```

This limiting method would grant access to the all users over the IP-range 257.0.0.1 to 257.0.0.255 and 192.168.*.* .

Starting the Apache Proxy

To start the process, copy the `httpd` startup script to `proxyd`. On SystemV and most other systems, the script is usually named `httpd`; however, on

Slackware the script is named `apachectl`. The necessary Slackware scripts are the following:

`/etc/rc.d/rc.M`. Run-level script that starts the service at boot time.

`/etc/rc.httpd`. Script that contains the command and arguments used to start the server.

`Apachectl`. Script used to start the server.

Next, edit the `proxyd` or the `apachectl` script using a text editor. Find the section labeled `START`. This section should look like the following before it is edited:

SYSTEMV SYSTEMS

```
start)
echo -n "Starting httpd: "
daemon httpd
echo
touch /var/lock/subsys/httpd
```

BSD SYSTEMS (SLACKWARE)

```
# the path to your httpd binary, including options if necessary
HTTPD=/var/lib/apache/sbin/httpd
```

After editing the files, restart the server. Remember that when restarting the server `httpd` and `proxyd` or `apachectl` and `proxyctl` both have to be started. After editing the configuration files, they should look like the following:

SYSTEMV SYSTEMS

```
start)
echo -n "Starting proxy: "
/usr/sbin/httpd -f /etc/httpd/conf/proxyd.conf
echo
touch /var/lock/subsys/proxy
```

BSD SYSTEMS (SLACKWARE)

```
# the path to your httpd binary, including options if necessary
HTTPD=/var/lib/apache/sbin/httpd -f /var/lib/apache/conf/proxyd.conf
```

NOTE **The `-f` option is used to specify the configuration file.**

Security Fundamentals

The visitors to a Web site have access similar to the user ID that is running the server process, which is fine because that user ID should have limited permissions. When things are configured properly, the user ID has a narrow range of functions it is allowed to perform on the machine. Keeping the system secure means keeping that user ID's access limited to a narrow range.

NOTE **Although Apache can be started by an individual without root access, the Apache process will not be able to `su` to a different user ID (therefore running as that user) and will not be able to attach to a port number less than 1,024. This can be useful under some limited circumstances but is not ideal for a publicly accessible system.**

Permissions

File permissions are an important aspect of security for a Linux system. If care is not taken with the permissions of certain critical files, regular users can implement changes that are normally restricted to the superuser.

For Apache to run as a non-privileged user on port 80 (or any port less than 1,024), the user must launch it from root. This non-privileged user (let's call it `webuser`) must be able to read the Web document tree (but probably not make changes to it). So that the `webuser` is able to read the document tree, be sure that all files and directories that must be served from the document tree are set world-readable:

```
# chmod -R o+r
```

Create a group for those who will be working on the Web documents (let's call it webeditors). This must be a different group from the one `webuser` belongs to, as it will be writing to the document tree. Setting the group sticky bit for the `<htdocs>` directory will ensure that any new files created there will be owned by the group and can be edited by any member of the group:

```
# chown -R otheruser.webeditors
# chmod -R g+s
```

The `-R` switch causes the operation to be performed recursively so that all directories below the current directory are affected by the `chown` and `chmod` commands.

NOTE **You can specify both the owner and the group name in the `chown` command by separating them with a period, which will save you from having to run a separate `chgrp` command to change the group ownership.**

To use directory indexes or to read the directory from a script or Web page, directories will have to be set world-executable:

```
# chmod o+x directory
```

A common problem with log files is that the Apache process is not able to write to the log files or log directory. If this is the case, a simple solution is to give ownership of the log file to the `webuser` account:

```
# touch /var/log/apache/{error,access}.log
# chown webuser /var/log/apache/{error,access}.log
```

The `touch` statement creates the log files; the second statement gives ownership to the `webuser` account. For additional protection (on newer file systems), you can also restrict the user to append-only access on the log files:

```
# chattr +a /var/log/apache/{error,access}.log
```

These commands are helpful when first creating the log files and should also be included in any log rotation scripts you install so that the attributes are maintained across log rotations.

Scripting

Running CGI scripts on Apache allows a user to create dynamic content. A drawback of this expanded functionality is that security holes can result if you do not correctly configure the server and verify that the CGI scripts are functioning properly.

The nature of scripts means they have to be interpreted with the access capabilities of some user, typically the same as the `httpd` process. Make

sure that this user is not inadvertently given the ability to make any undesired changes to the system.

A certain amount of trust is required of scripts and those who have written them. Because many CGI scripts can be rather complicated, evaluating them for problems can be difficult. Many scripts are available on the Internet for download, and some of the authors might have motives that go beyond providing you with a feature for your server. Similarly, an author can write a script without full knowledge of the repercussions of the security issues involved, leaving the server open to attack by someone familiar with the failings of that script.

WARNING **Never place interpreters in the `cgi-bin` directory. This applies to Perl, PHP, or any other language. This is usually a problem with the Windows environment, but Linux users need to be aware of it.**

suExec

One problem with CGI scripts is that they are run by the owner of the server, which can cause serious problems if the script is written in a malicious or careless way. Typically, the damage is restricted to whatever the server processes' owner may access. Under some circumstances, a visitor can enter data that causes a script to do things the author didn't intend, or a visitor who has access to the system can modify a script to do things beyond the intentions of the original author.

The `<suexec>` option allows CGI scripts to run under a user ID other than the owner of the server and, at the same time, applies a series of strict tests to what the script is trying to do. If a script tries to step out of the prescribed parameters, it is terminated and an error is logged to the `<suexec>` log file.

The Apache documentation warns that `suexec` should be used only by someone who is familiar with the `suid` and `sgid` processes. An improper implementation by an inexperienced administrator could result in unexpected actions by the script.

NOTE **Configuring `suexec` is beyond the scope of this course. If you would like to experiment with it, read the server documentation carefully, paying close attention to permissions and ownerships.**

Matrix of Ideal Permissions

Permissions can be set to read (`r`), write (`w`), or execute (`x`). Users require various levels of permissions, in order to both accomplish their tasks and keep the system secure. Not every user should have permissions to every file. Table 4.3 offers guidelines for setting the permissions of different groups.

NOTE **Logs also include the lock and state files created by Apache under `/var`.**

User Access Control

In the past, administrators used the `access.conf` file to define locations that the world could access on the server machine. It is still available for legacy purposes, but its directives have all rolled into the `httpd.conf` file. Although it is not needed, `access.conf` may be used. In our examples, we use the `httpd.conf` file for all directives, including those for access control.

Common Access Controls

The following represent some general access directives that affect the server as a whole.

DirectoryIndex index.html

This directive defines the default index page, the file that will be loaded when a directory is accessed without specifying a file. Sites on the Internet on which the main page looks like an FTP directory usually are missing the index file defined for this directory. `index.html` is often used, but other names are also common. If a list of file names is specified and files matching several of the names are present, the first file on the list is the one that Apache uses.

Table 4.3 Matrix of Ideal Permissions

GROUP	CONFIGURATION	TOOLS	LOGS	CGI	DOCUMENTATION
Webmaster	rwx	r-x	r-x	rwx	rwx
Web developer	---	---	---	rwx	rwx
Web author	---	---	---	r-x	rwx
Web server	---	---	---	r-x	r-x

AccessFileName .htaccess

When the `AllowOverride` directive is allowed in a directory, some access methods can control the settings for that directory. This directive sets the name for that file to `.htaccess`.

Blocking Access

It is possible for the outside world to view the entire contents of a server machine's file system if there are improper symbolic links to content that is accessible from outside the server. In other words, if an innocuous `index.html` file were linked to `</>`, everything on the system would be available for perusal.

The `<Directory>` directive can prevent mistakes. It allows a user to apply rules exclusively to a particular directory. By default, the directory will inherit the preexisting directives on the server.

To control access to your Web content, there are a number of directives at your disposal:

```
Deny and Allow
Order
Deny and Allow
```

These directives use the visitor's originating host as a criterion to determine if access will be permitted. As with most Linux/Unix security models, these criteria are based on one of two perspectives that specifically denies or allows.

The syntax is as follows:

```
Deny from host
```

or

```
Allow from host
```

The host specified can be a specific host name, an IP address, or any of the following criterion arguments:

`all`: Wildcard, all hosts.

`domain-name`: Exact or partial names will be matched.

`IP address`: Exact or partial addresses will be matched.

network/netmask pair: Range of both will be used.

CIDR specification: A different way of specifying network/netmask, such as 192.168.0.0/16.

Order

Order simply tells Apache the order in which it should handle a set of `Deny` and `Allow` directives below it. One of the following three will be used for an order directive:

deny,allow Evaluate the deny directives first.

allow,deny Evaluate the allow directives first.

mutual-failure Hosts must be specifically allowed while not matching any deny criteria.

The `Satisfy` directive can be used in concert with the `Allow` and `Deny` directives. It is useful only if access to a directory, location, or file is restricted by both username/password and client host address. When this happens use the `Satisfy` directive to modify the default behavior (which is to require both). Within the directory or `.htaccess` context, `Satisfy` means that the client only has to satisfy host restriction or enter a valid username and password.

Do not put spaces between the `deny` and `allow` keywords. If you include spaces, Apache will fail to start, displaying a configuration error. The following is correct:

```
Order deny,allow
```

while this is incorrect:

```
Order deny, allow
```

`Deny` and `allow` work together by evaluating the originating host according to what you chose to deny and allow and then granting access depending on the outcome.

First, block access to everything:

```
<Directory />
  Order deny,allow
```

```
  Deny from all
</Directory>
```

Now that everything is blocked, we can follow through and let in what we want. The following represents a server that has all of its content in one directory, `/www/htdocs`:

```
<Directory /www/htdocs>
  Order deny,allow
  Allow from all
</Directory>
```

Sometimes an allowed directory will have files inside that need to be blocked. Implementing a `<Files>` container can take care of this. For example, the `.htaccess` file contains information about local changes to the configuration and may be of interest to prying eyes.

Here is an example using a `<Files>` container to block access to a file:

```
<Files .htaccess>
    Order allow,deny
    Deny from all
</Files>
```

As of Apache 1.2, a pattern to block any matching files can be specified; however, the `FileMatch` directive is the preferred method for pattern matching in Apache 1.3. Also note that a `Files` container can be nested inside a `Directory` container.

Enabling Content from Home Directories

In many environments, such as corporate intranets or university installations, individual users on the system are interested in serving Web content. One way to give individuals the ability to serve their own content is to provide a place within each user's home directory.

Because access has been blocked to the root file system in the preceding definition, the risk associated with having content in the home directories is reduced. The main directives that allow system users to provide content are as follows.

UserDir html

System users can create a directory in which any content they place will be served. This directive defines the name of the directory within the user's home directory. A URL such as www.xyz.com/~joe would go to `/home/joe/html`, the specified directory within joe's home directory.

The following directives define some rules that are directory specific, so they are bracketed within a `<Directory>` section. The general form is:

```
<Directory />
    Options
    AllowOverride
</Directory>
```

More specifically, here is an example of the first directive:

```
<Directory /home/*/html>
```

This directive appears again as we specify where the home directories exist. It also marks the beginning of the section that contains the special directives that are needed.

Here is an example of the second directive:

```
Options Indexes +SymLinksIfOwnerMatch IncludesNoExec
```

These turn on a set of options in a section. Choices can be `None`, `All`, or any combination of `Indexes`, `Includes`, `FollowSymLinks`, `ExecCGI`, or `MultiViews`. The `MultiViews` option must be explicitly named because it is not included in `Options All`. Note that the `FollowSymLinks` option, while helpful, could allow a user to accidentally create a symbolic link to an area that should not be visible to visitors, such as `/etc`. The `SymLinksIfOwnerMatch` option allows the use of symbolic links without opening up this potential security leak.

Here is an example of the third directive:

```
AllowOverride all
```

When multiple directories are being defined, a file called `.htaccess` (named in the `AccessFileName` directive) can be used within each directory to provide further control of the directory. The user may or may not

want this file to be able to change things that have been defined as system defaults. This directive can use wildcards, such as `All` or `None`, or can specify individual overrides, such as any combination of `Options`, `FileInfo`, `AuthConfig`, and `Limit`. When override is permitted, Apache searches each level of the directory tree containing the current directory for an instance of the access file. When the directory tree of the current directory is deep, this can impose significant performance penalties, which is why an `AllowOverride` none restriction is commonly imposed on the root directory. Override is enabled only for specific directories where override is expected to be used.

An example of the directive that will close the `<Directory>` section is:

```
</Directory>
```

Access Directives

Password-protected content needs to have a special definition that will provide information that both Apache and the Web browsers will use while negotiating the connection to the resource.

AuthType

This should be set to `basic`. `Digest` could also be placed here to enable encrypted passwords, but not all browsers will support that feature.

AuthName

Password-protected areas are typically a directory and all subdirectories within that directory. This section of content is known as a realm. Users who pass the password challenge are allowed access to content within the realm.

AuthName "Private Documents"

Users must not use their system password for their Web password because cracking the Web password is much easier and has almost none of the protections that even the simplest system password arrangement has. The following are directives listed within a directory's `.htaccess` file and/or `httpd.conf <Directory>` section. They can implement restricted use for a directory containing important information.

AuthGroupFile. This file associates group names with their members. The file can be edited by hand, with each line representing a unique group and its associated users. This is optional unless you wish to use group authorization.

AuthUserFile. Points to the file that contains usernames and passwords. Use the `htpasswd` utility to create entries in your file because the passwords need to be in an encoded format. The `/etc/passwd` file is not to be used for the `AuthUserFile`. A separate user authorization file will need to be created.

Require. This directive is used to specify who can access restricted content. Whoever is included in the `Require` directive will have the opportunity to enter a password. Arguments can include a list of users and/or groups. Those who meet the criteria defined here will have an opportunity to enter a valid password to gain access.

NOTE **The files to which `AuthGroupFile` and `AuthUserFile` refer must be outside of the directory tree and defined in `DocumentRoot`.**

Defining within httpd.conf

To use an entry with `httpd.conf` to control all aspects of the authorization resource, make a directory entry that contains the relevant information. An entry for a working authorization setup is as follows:

```
<Directory /www/htdocs>
  AuthType Basic
  AuthName "Secret Place"
  AuthUserFile /www/users
  AuthGroupFile /www/groups
  Require group kernelgroup
</Directory>
```

This entry creates a protected space on the server within the `/www/usr/apache/htdocs` directory. As it is also the `DocumentRoot` of the server, Apache requests a password before it provides any content from the server. Group and password files were created then defined so that `mod_auth` could find them. Finally, the `Require` directive has been instructed to authorize members of the `kernelgroup` only.

When loading a page within `/www/htdocs`, a user will receive a password prompt. If the user types in a username that is a member of

the `kernelgroup` and can provide a password that matches the username, the user can enter.

Defining within Control Files

Another way access can be restricted is to use a file within the main directory of the content itself. Apache looks for the file and then uses it for directory-specific configuration. A default may already exist for this; if not, add this line to the `httpd.conf` file to specify it:

```
AccessFileName .htaccess
```

Files with the name `.htaccess` that reside within content directories will be used to control access within those directories. Here is the control file that has been placed within `/www/htdocs` used to provide the same protection as the previous example:

```
#.htaccess
AuthType Basic
AuthName "Private Stuff"
AuthUserFile /www/users
AuthGroupFile /www/groupsfigure
Require group kernelgroup
```

As before, the users within `kernelgroup` have an opportunity to provide a valid password.

Shells and Commands

There are many ideas and concepts behind file security, especially over a network. This section will describe some of these concepts and discuss how they are used in Apache.

Checksums

A checksum is a computed value that depends on the contents of a block of data and that is transmitted or stored along with the data in order to detect corruption of the data. The receiving system recomputes the checksum based on the received data and compares this value with the one sent with the data. If the two values are the same, the receiver has some confidence that the data was received correctly.

The checksum may be 8 bits (modulo 256 sum), 16, 32, or some other size. It is computed by summing the bytes or words of the data block, ignoring overflow. The checksum may be negated so that the total of the data words plus the checksum is zero.

PGP and Checksums

PGP uses a checksum value to do the following:

- Produce noncryptographically secure pseudo-random numbers. A strong pseudo-random number generator (PRNG) exists for the production of numbers that need cryptographic security.
- Check whether a message has been corrupted during transit.

NOTE **This is in addition to any cryptographically secure method of error detection.**

Using a checksum and PGP helps to ensure that what you are downloading is actually the program you think it is. Otherwise, you could be downloading a Trojan horse or some other malicious program. OpenPGP (v1) does not mandate that conventionally encrypted non-signed messages are error checked, so errors may exist without warning. The checksum function is used only in areas that don't require cryptographic strength. When cryptographic strength is required, PGP uses a hash function.

To perform a checksum, type the following on the command line:

```
# md5sum file_name
```

The output will look something like this:

```
55596d03cf27c88ca0614aad016026ae  file_name
```

What Is MD5?

MD5 reads data and calculates a cryptographic checksum that is very hard to duplicate. Just as traditional checksums give assurance that a file has not been accidentally modified, MD5 ensures that a file has not been intentionally modified, which helps to protect from downloading any Trojan horses.

Currently, no one knows how to modify a file without changing its MD5 checksum. Researchers continue to try and are making some progress toward the eventual goal of breaking MD5, but it is still considered strong enough for most uses.

Password Authentication

Apache can perform authentication tasks through the `mod_auth` module. It is not necessary to specify its inclusion during compilation, as it is part of the default configuration. A handful of directives are needed to implement authentication with `mod_auth`. These directives work in conjunction with the `.htaccess` file or `<Directory>` section to control access.

For authentication to work, you must provide `mod_auth` with the information it needs about the users and the content that is to be protected, including the following:

- User and group information
- Access directives

User

Information about the users is stored in a special file that you can create for a given Web resource. This file requires encoded passwords, and a utility called `htpasswd` exists to generate such entries:

```
htpasswd -c filename username
```

The `-c` tells `htpasswd` to create a new password file. If there is already a file by that name, the `-c` is not necessary; otherwise, the password file will be rebuilt and then truncated. Here is an example:

```
# htpasswd -c /www/passfile tux
```

After invoking `htpasswd` with the desired arguments, the user will be prompted to enter a password. The username will then be added with its encoded password.

Group

A file containing group information is useful to authorize several users at once. Being an authorized user does not guarantee access to the protected

content because a valid password is still required. The name of the group file is unimportant as long as it exists and the `AuthGroupFile` directive can find it. The user can edit this file manually. Its format is as follows:

```
groupname: member1 member2 member3
```

System Utilities

In addition to the internal configurations that can be compiled into Apache, there are a number of utilities that will help with keeping an Apache install secure. Some of these utilities are discussed in the following subsections.

Server-Side Includes

A `Server-Side Include` directive appears inside an HTML document, within comment brackets, like this:

```
<!-- #directive arguments -->
```

Basic Commands

Use SSI directives in HTML documents to echo information or any environmental variable:

```
<!--# echo –Hello!Ó -->
```

Variables

SSI has any environmental variable at its command:

```
<!--#echo var=DATE_LOCAL -->
```

SSI can modify these variables. In this example, `timefmt` is a string used when printing dates:

```
<!--#config timefmt=Ó%AÓ -->
Today is <!--#echo var=DATE_LOCAL -->
```

If today were Wednesday, it would say:

```
Today is Wednesday
```

The `%A` tells SSI to display just the full name of the day of the week. Table 4.4 shows other format commands you can use. They are the same output controls used for the Unix date function.

Table 4.4 Format Commands

COMMAND	DESCRIPTION
`%%`	A literal %
`%a`	Locale's abbreviated weekday name (Sun..Sat)
`%A`	Locale's full weekday name, variable length (Sunday..Saturday)
`%b`	Locale's abbreviated month name (Jan..Dec)
`%B`	Locale's full month name, variable length (January..December
`%c`	Locale's date and time (Sat Nov 04 12:02:33 EST 1989)
`%d`	Day of month (01..31)
`%D`	Date (mm/dd/yy)
`%e`	Day of month, blank padded (1..31)
`%h`	Same as %b
`%H`	Hour (00..23)
`%I`	Hour (01..12)
`%j`	Day of year (001..366)
`%k`	Hour (0..23)
`%l`	Hour (1..12)
`%m`	Month (01..12)
`%M`	Minute (00..59)
`%n`	A new line
`%p`	Locale's AM or PM
`%r`	Time, 12-hour (hh:mm:ss [APJM])
`%s`	Seconds since 00:00:00, Jan 1, 1970 (a GNU extension)
`%S`	Second (00..60)

Table 4.4 *(Continued)*

COMMAND	DESCRIPTION
`%t`	A horizontal tab
`%T`	Time, 24-hour (hh:mm:ss)
`%U`	Week number of year with Sunday as first day of week (00..53)
`%V`	Week number of year with Monday as first day of week (01..52)
`%w`	Day of week (0..6); 0 represents Sunday
`%W`	Week number of year with Monday as first day of week (00..53)
`%x`	Locale's date representation (mm/dd/yy)
`%X`	Locale's time representation (%H:%M:%S)
`%y`	Last two digits of year (00..99)
`%Y`	Year (1970…)
`%z`	RFC-822 style numeric time zone (-0500) (a nonstandard extension)
`%Z`	Time zone (e.g., EDT), or nothing if no time zone is determinable

XSSI

Apache goes beyond SSI with eXtended SSI (XSSI). With XSSI users can attach conditions to the execution of an SSI directive, define variables, and execute external programs and CGIs (if security allows it).

To create a variable with XSSI use the following:

```
<!--#set var="city" value="Carthage" -->
```

Shell Commands

XSSI directives can be used to execute shell commands, unless the directive is turned off in `httpd.conf`:

```
<!--#exec cmd="ls" -->
```

would list all files and directories in the directory of the file being served.

This will produce something similar to Figure 4.3.

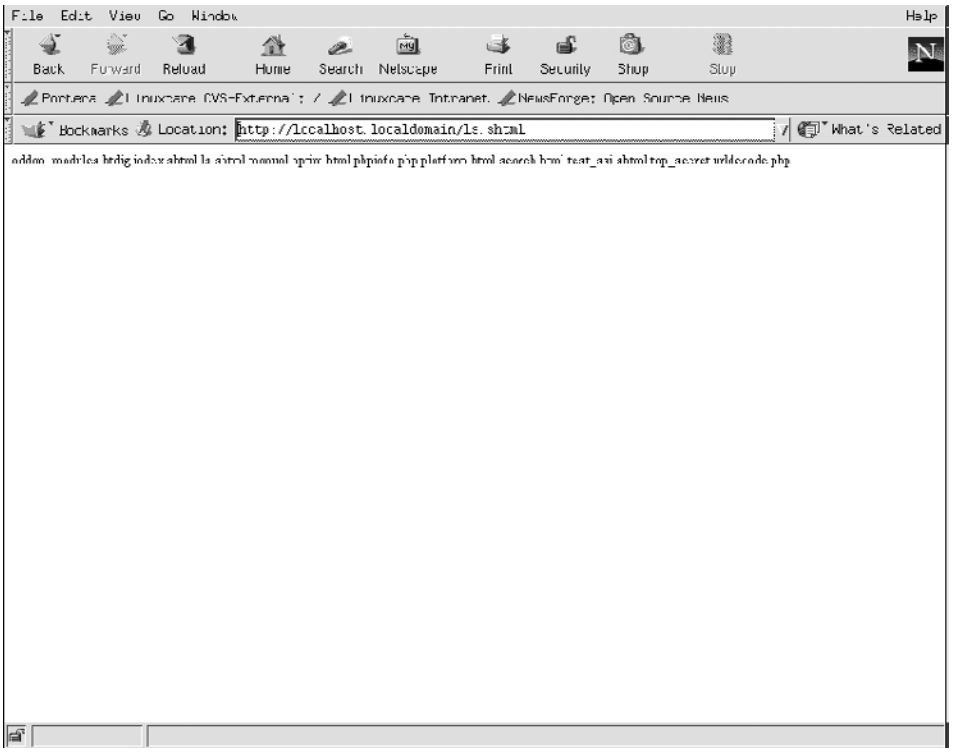


Figure 4.3 Screenshot of a directory listing displayed in a Web browser using the ls command.

Users can also employ the `exec` command to execute CGI scripts easily, as in the following example:

```
<!--#exec cgi="hello.pl" -->
```

The `exec` option can be turned on and off in the `httpd.conf` file with the `Options` directive, which means users have control over where SSI is allowed to execute files. Look for the `Options` directive and see if it has the `Includes` option or the `IncludesNOEXEC` option. The `Includes-NOEXEC` option allows you to use `Server-Side Includes` but does not allow you to use the `exec` command.

Including Files

XSSI can include files and execute external CGI scripts or shell commands.

Use the file or virtual attribute to get different results. For example:

```
<!--#include file="base.html" -->
```

is different from

```
<!--#include virtual="/base.html" -->
```

Using the file attribute is a little less useful. It must be a file path relative to the current directory, and it can not start with `/` or `../`. The virtual attribute specifies a URL relative to the page in which it is served, and it must be on the same server.

Executing Scripts

Because any file using SSI can be included, the output from CGI scripts can also be included. For example, the following:

```
<!--#include virtual="/cgi-bin/cool_hit_counter.cgi" -->
```

would include the preceding `cgi` in the HTML document.

Embedding XSSIs

An important note is that includes can be embedded. For example, imagine that a footer is used for files residing deep in a Web site's directory structure, but it needs to access the `cool_hit_counter cgi`. Two includes can be in every file:

```
<!--#include file="footer.shtml" -->

<!--#include virtual="/cgi-bin/cool_hit_counter.cgi" -->
```

To change CGIs and to move to the `groovy_hit_counter.cgi`, the include in every file must be changed. An alternative is to have the virtual include exist only in `footer.shtml`, which will make upgrading hit counters less troublesome.

Conditional Statements

XSSI supports conditional statements, such as `if`, `else`, `elif`, `&&`, and `||`. These statements allow the developer to customize Web content without the use of a full-fledged CGI.

For example, in the following HTML file, the day is checked and a custom message is displayed based on the result:

```
<!--#config timefmt="%A" -->
<!--#set var=today value="DATE_LOCAL" -->
<!--#if expr="$DATE_LOCAL = /Friday/" -->
      <h2 align="center"><font color="#FF0000">IT IS FRIDAY!!!</ font></h2>
<!--#elif expr="$DATE_LOCAL = /Monday/" -->
      <h6 align="center">Please, just one more cup of coffee</h6>
<!--#else -->
      At least it isn't Monday
<!--#endif -->
```

Notice the use of the `$DATE_LOCAL` variable, which is an environmental variable. Using the following command provides a list of the environmental variables available to XSSI:

```
<!--# printenv -->
```

This will produce something similar to Figure 4.4.

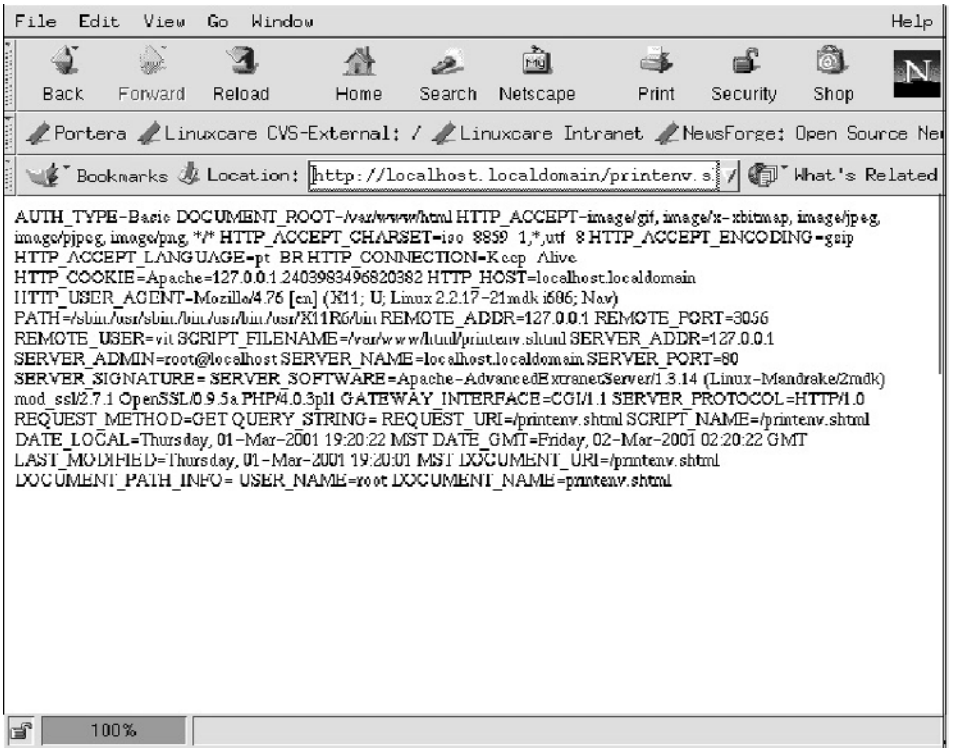


Figure 4.4 Screenshot of the XSSI Printenv command.

To set a variable in XSSI, use the `set` command:

```
<!--#set var="boat" value="Titanic" -->
```

Now, whenever `echo boat` (`<!--#echo var="boat"-->`) is used, XSSI will return `"Titanic"`.

For another reference, see Apache's SSI tutorial at http://httpd.apache.org/docs/howto/ssi.html.

ModSSL versus Apache+SSL

There are two main SSL packages that can be used in an Apache install, ModSSL and the Apache+SSL patch. The following section presents the advantages, disadvantages, and installation issues of both packages.

SSL—mod_ssl

`mod_ssl` is a module derived from Ben Laurie's Apache+SSL patch.

Essentially, these patches are given to the community at large, as alternatives to one another. They are no different from choosing whether to use Vi or Vim.

The Secure Sockets Layer (SSL) protocol was developed by Netscape to provide a secure channel for sensitive information, such as that used in online credit card transactions. SSL is now an Internet standard, and Apache is able to implement it through a module.

SSL—Secure Sockets Layer

The Secure Sockets Layer protocol is a protocol layer that may be placed between a reliable connection-oriented network layer protocol (e.g., TCP/IP) and the application protocol layer (e.g., HTTP). SSL provides secure communication between client and server by allowing mutual authentication, the use of digital signatures for integrity, and encryption for privacy.

The SSL protocol is designed to support a range of choices for specific algorithms used for cryptography, digests, and signatures. This allows algorithm selection for specific servers to be made based on legal, export, or other concerns, and it also enables the protocol to take advantage of new

algorithms. Choices are negotiated between client and server at the establishment of a protocol session.

The SSL protocol is a low-level authentication and encryption method used to secure transactions in higher-level protocols, such as HTTP and FTP. It was developed by Netscape, but support is provided by most browsers and Web servers. The current SSL protocol version is 3.0, but the IETF Transport Layer Security (TLS) specification Version 1.0 actually supersedes it. (SSL and TLS are generally discussed together, as TLS is mainly a refined version of SSL.)

SSL includes provisions for server authentication (verifying the server's identity to the client), encryption of data in transit, and optional client authentication (verifying the client's identity to the server). By employing SSL-enabled servers and clients, it is possible to send encrypted data, such as passwords and credit card numbers, across the Internet without fear of interception.

Clients requesting documents stored in SSL-enabled directories must use the `https://` URL format instead of the standard `http://` format. One reason for this is that SSL-enabled Web services are generally offered on a different port than nonsecured transmissions, normally port 443.

Implementing SSL in Apache

There are two ways to add SSL to Apache. The first method is to replace Apache with Apache-SSL. This is a version of Apache that has SSL security included. (Actually, it is distributed as a patch to the Apache source code.) It is available at the www.apache-ssl.org site. The other way is to add the `mod_ssl` module to a standard version of Apache. (In most circumstances, Apache must be recompiled to make `mod_ssl` work.) More information on `mod_ssl` can be found at www.modssl.org/. We will focus on Apache-SSL for the rest of this course.

Legal Issues

SSL data transport requires encryption, and many governments have restrictions on the import, export, and use of encryption technology. Also, a few of the encryption algorithms are patented in some countries. With SSL included in a software package, its distribution involves a number of legal issues. This is why SSL is distributed separately from the standard Apache distribution.

Until recently, the United States prohibited the export of strong encryption, normally defined as anything beyond 40 bits. This ban has recently been eased, but the full implications of the new law have not been entirely analyzed. Within the United States, SSL can be used for any purpose due to the expiration of the patent held by RSA Data Security on the RSAREF code. This patent never applied generally outside the United States, so now you should be able to use the code for any purpose.

Even though your Apache server should be well established by now, it will require constant maintenance. The next chapter will discuss some simple solutions to common problems that may arise.

CHAPTER 5

Troubleshooting

Objectives

- List Apache online troubleshooting resources.
- List common Apache messages and debugging options.
- List common configuration and logging problems.

Online Troubleshooting Resources

The online Apache manual at www.apache.org/docs/ is always a good place to start. There is also a generic Apache FAQ available at www.apache.org/docs/misc/FAQ.html.

Search the live Problem Report (PR) database at http://bugs.apache.org. Near the top of this page you can enter a search string and search the entire database. Further on down the page you can enter a number of parameters to narrow your search.

At http://dev.apache.org you will also find links to archives of the new-httpd mailing list. This archive is the list that the developers of Apache have been using for years to discuss issues related to the development of the Web server. There is a lot of information in these archives. Subscribing to the new-httpd mailing list just to listen is a good idea. Subscription instructions along with a list of other available mailing lists can be found at: http://dev.apache.org/mailing-lists.html.

The primary user support forum available for Apache is the comp.infosystems.www.servers.unix newsgroup on USENET. Archives of this group can be found at www.deja.com/group/comp.infosystems.www.servers.unix.

Tracking Down an Apache Core Dump

Occasionally, things can go wrong. There are many reasons for Apache crashing, such as a bug in the Apache code, bad memory in the machine, or a bug in a third-party module that was added. In order to understand what might have caused the crash, there are a number of things that can be done.

Check the `/var/apache/logs/error_log` file.

Here are some possible messages:

```
Tue May  2 09:20:46 2000] [notice] Apache/1.3.14
(Unix) mod_perl/1.21 configured — resuming normal operations
```

This is a good message. It means the server has started up and that it is happy.

```
Tue May  2 09:21:17 2000] [notice] caught
SIGTERM, shutting down
```

This is also a good message. It means the server was shut down normally using something such as `apachectl stop`.

```
Tue May  2 09:41:25 2000] [notice] SIGHUP
received. Attempting to restart
```

This is a good message. It means the server was restarted with a command, such as `apachectl` restart.

```
Tue May  2 09:31:14 2000] [warn] pid file
/var/run/httpd.pid overwritten — Unclean
shutdown of previous Apache run?
```

This is not a very good message. It means that something happened to the Apache server on its last run and that it did not shut down cleanly.

```
[Tue May  2 09:31:44 2000] [notice] child pid
10187 exit signal Segmentation Fault (11)
```

This is a very bad message. It means one of the httpd child processes core dumped.

If a `Segmentation Fault` error or perhaps a `Bus Error` is found in the `error_log` file, there are a couple of tools available that might provide a hint as to what might be causing the problems:

```
/opt/sfw/bin/gdb
/usr/bin/truss
/usr/bin/mdb
```

A typical use of `gdb`:

```
# gdb /usr/apache/bin/httpd
GNU gdb 4.18
Copyright 1998 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are welcome
to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
You can compile the httpd from the Source Package.
PATH=/usr/sbin:/usr/bin:/opt/sfw/bin:/usr/ccs/bin
cd /usr/share/src/apache
```

NOTE **We run httpd with the -X flag, which means that it will be started in non-forking mode. This makes it much easier to trace, as we have only a single httpd process to worry about this way.**

Some Useful Sites

The following sites will help you in your troubleshooting efforts:

cronolog (www.ford-mason.co.uk/resources/cronolog). A log file rotation program for the Apache Web server.

ApacheWeek (www.apacheweek.com). A free online magazine with tips.

O'Reilly Network Apache Forum (www.oreillynet.com/cgi-bin/conf/summary?group=oreillynet.apache). A discussion forum for Apache users.

Devshed Forums (www.devshed.com). A discussion forum that covers a multitude of Apache problems and solutions.

Configuration Issues

Problems in configuration may cause Apache not to start at all, or they may cause requests to fail or to be served other than the intended way.

Permissions problems occur because of several security principles involved on administering a Web server:

Root. The system user allowed to bind low-numbered ports and access all files.

httpd or nobody. Who the Web server "runs as."

Webmaster. An administrator or other users who are able to start and stop the Web server.

remote user. Those who are able to read.

author user. Those who create content but do not administer the Web server.

Logging Problems

With multiple servers running on a single instance of the Web server daemon and separate log files for each server, the user may become aware of the consumption of file descriptors (also called file handles). If the error logs start reporting such errors as unable to fork or that access logs are not being written, this has happened. Basically, the problem is that Apache uses a file descriptor for each distinct log file and 10 or 20 for internal use. Some systems have a limit of 64 file descriptors for a process, which may usually be

increased up to a much larger number. Apache will attempt to increase the limit as needed but may have problems under certain circumstances:

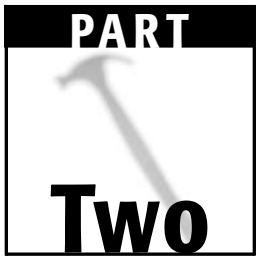
- If the system does not support the setlimit() call (or if it does not work, as on Solaris 2.3)
- If the number of descriptors exceeds the absolute limit

If one of the first two conditions applies, starting Apache with a script similar to this can help:

```
#!/bin/sh
ulimit -S -n 100
exec httpd
```

An alternate solution is to send all logs to the same file and filter them out later for distribution to each client. A description of real-time logging using piped logs is at www.apache.org/docs/new_features_1_3.html.

This issue and other problems are addressed in detail at the Apache Web site, www.apache.org.



Labs and Exercises

LAB

1

Installation

Purpose

In this exercise, you will learn how to install and update Web server modules, features, and configurations so that they can meet your needs.

This exercise will be centered around four packages that you will install: the `apache-1.x.x.i386.rpm`, `php-4.0.x.x.i386.rpm`, `mod_php-4.x.x.i386.rpm`, and `mod_perl-1.x.i386.rpm` files. The installation of these modules will give you moderate Web server functionality.

After these are installed, you will also need to complete any site-specific configurations to the Web server that are required. The files used in this part of the exercise will commonly be contained in the `apache/conf` directory, with most of the work centering on the `httpd.conf` directory.

Theory

For this exercise, we are using the `.rpm` files for the installation/update. The `.rpm` files are commonly used among many different distributions, not just Red Hat; however, a few distributions, including Debian, Slackware,

and Corel, do not include `.rpm` files in their distribution media, such as CD-ROMs. Even though `.rpm` files are not available in these distributions, you will often find that the rpm program is included. This program allows the user to install `.rpm` files, even if they are not included on the distribution's CD.

Lab Exercises

This lab will go step by step through the installation, from downloading the required packages to getting them installed and configured. The lab will include the following steps:

1. Downloading modules.
2. Preinstalling Query.
3. Installing the package.
4. Setting up the basic server.

At the completion of this lab, the server should have the basic functions needed.

Downloading Modules

The first step of this exercise is to acquire the packages needed. This can be done in many ways, including distribution media installation and acquisition from the Internet. The two most common places to find the packages are distribution CDs or the Internet. The packages on distribution CDs are not always up-to-date, and the desired packages are not always readily available. It is much easier to use packages downloaded from the Internet because they offer a centralized location where the packages can be found. This makes the packages very easy to find and readily available.

Finding the `.rpm` packages on the distribution's CD is easy. First, make sure the media is mounted. Then, enter the directory structure and locate the `.rpm` files. These are usually located in a directory that shares its name with the distribution. For example, the command for installing a package on a Red Hat CD would most likely be this:

```
bash# rpm -i /mnt/cdrom/redhat/RPMS/packagename.ver.rpm
```

For SuSE, the files are located in the subdirectories of the `/suse` directory, as in the following example:

```
bash# rpm -i /mnt/cdrom/suse/sd/packagename.rpm
```

The location of the packages varies between distributions, and it can take some time to locate the packages you desire. The most up-to-date packages must be obtained from the Internet.

To find packages on the Internet, you must first locate a reliable site. RPMfind (www.rpmfind.com) and the official Red Hat Web site (www.redhat.com) are two of the most reliable sources for finding `.rpm` packages.

The packages that you will need for this lab are these:

```
apache-1.x.x.i386.rpm
mod_perl-1.x.i386.rpm
php-4.x.x.i386.rpm
mod_php-4.x.x.i386.rpm
```

The `x`'s represent the available, or current, minor version number of the package.

Once you have these packages, make sure they contain what you will need and do not contain errors.

Preinstallation Query

Once you have the packages, make sure that the packages are legitimate and do not have any visible errors. To do this, you will have to run the `rpm` command to verify that the acquired packages are valid. This is done by running `rpm` with the `-qp` argument, which allows you to query the uninstalled package. The following is an example of how to query the uninstalled Apache package:

```
bash# rpm -qp apache-1.3.12-25.i386.rpm
```

If nothing is wrong with the package, the command will return the name of the package contained in `apache 1.3.12-25.i386.rpm`, as in the following:

```
bash# rpm -qp apache-1.3.12-25.i386.rpm
apache-1.3.12-25
```

This means that the `apache-1.3.12-25.i386.rpm` package contains the `apache-1.3.12-25` package. Repeat this process with all of the packages that you are going to install. The next step will be to install the packages and then to verify them.

Package Installation

To install this package, use the `-i` option. It will install the selected package, as shown:

```
bash# rpm -i apache-1.3.12-25.i386.rpm
```

This installs the file that has been identified as the Apache package. Once this package is installed, continue to install all of the other packages in the same fashion. After the packages are installed, you will have to verify them to make sure that they were installed properly, without errors. To do this, run `rpm` once again; this time use the `-V` option and the name of the installed package. The following command will test the integrity of the Apache package that was just installed:

```
bash# rpm -V apache-1.3.12-25
```

If the package has returned output, then errors are in the installation, and you should refer to the `rpm` man page in the signatures section; however, if the command did not return any output, then the package has been verified and should be safe to use. After all the modules are installed and verified, you may wish to view the files installed by the packages. This can be done by using the `rpm` command in the following manner:

```
bash# rpm -ql apache-1.3.12-25
```

This will display the list of files that are part of the specified package, in this case `apache-1.3.12-25`. The displayed output will be quite long, and you will not need to see many of the results. For most of this exercise, you will want to focus your attention on the files in the `httpd/conf` directory to customize the installation.

Basic Server Setup

Now that the verification and installation has been completed, the next step should be to edit the configuration files to customize Apache. This step is very easy if the administrator knows what to change. If you are new

to Apache, you might be wondering how to find documentation on editing the configuration files; the answer is surprising. Most of the required documentation is found in the configuration file itself, in the form of more than 650 commented-out statements. Most of these statements contain hints and instructions on how to set up the server. Without all of the comments, the file is only about a third as long.

Now, changes toward customization will take place. Open the `httpd.conf` file in Vi or the editor of your choice. The configuration depends greatly on what the Web server is going to do. It might be a good idea to uncomment the lines containing the modules that were installed. This can be done by finding the dynamic shared object (DSO) header in the `httpd.conf` file and changing the paths of the files in the following lines to where the modules are located:

```
LoadModule perl_module        modules/libperl.so
#LoadModule php_module        modules/mod_php.so
LoadModule php3_module        modules/libphp3.so
```

`LoadModule` is the directive to show that a module is being loaded, and it is followed by the module name and path. If the modules for the packages you just installed are in another directory, include the full path here.

Most of the other changes to make in the server configuration will be done simply by uncommenting lines. First determine what you want the server to do. After this is established, the needed changes can be made and the desired effect can be achieved.

Questions

1. How would an administrator keep clients from making more than one request per connection?
2. How would you find out what package is contained in a .rpm file?
3. If there is no DNS name for your machine, what should the server name be?
4. Where are the best places to find RPM modules?

Answers

1. Modify the `KeepAlive` directive so that it is turned off.
2. Use the `-qp` argument on the rpm command. (`rpm -qp <package>`).
3. The machine's IP address.
4. The sites of official vendors.

Advanced Questions

1. What would be a reason to edit a configuration file (other than `httpd.conf`)?
2. Describe the advantage of using DSO modules.
3. Describe the advantage of using static modules.

LAB

11

Install Apache+SSL

Purpose

The purpose of this lab is to install the Secure Socket Layer (SSL) protocol for an Apache Web server. With SSL, Apache offers the high level of security necessary for the safe transmission of sensitive data. With more and more businesses performing online transactions, security has become necessary to the continued growth of the Internet.

This lab will use an Open Source patch of SSL that is available from the Apache+SSL project and provides a secure connection between the server and the client's Web browser. This process will provide an interface between Apache and SSL.

Theory

SSL provides secure communications between a client and a server, allowing for mutual authentication, use of digital signatures, and encryption of communications. SSL is a low-level authentication and encryption method used to secure transactions for upper-level protocols, such as HTTP and FTP. Originally developed by Netscape, SSL includes provisions for server

authentication, encryption of data in transit, and client authentication. Using these methods, it is possible to send encrypted data, such as passwords and credit card numbers, across the Internet with a large degree of safety.

Due to various export restrictions, Apache and SSL are distributed separately; therefore, SSL is installed using a patch. Before using Apache+SSL, legal compliance should be checked.

NOTE **Everything from here forward will be done in the working directory `/usr/src/apache+ssl`.**

Lab Exercises

This lab will cover the following steps:

1. Downloading the latest release of the Apache server.
2. Compiling Apache with mod_ssl support.
3. Verifying that Apache was compiled with `mod_ssl`.
4. Testing the sample page in a Web browser.

Downloading the Apache server

As root, create the directory `/usr/src/apache+ssl` using the following:

```
$ mkdir /usr/src/apache+ssl
```

From the sites www.open-ssl.org, apache.org, and apache-ssl.org, download `openssl-0.9.6a.tar.gz`, `apache_1.3.14.tar.gz`, and `apache_1.3.14+ssl_1.42.tar.gz`, respectively. These files should be placed in the `/usr/src/apache+ssl` directory as shown here:

```
$ls
apache_1.3.14+ssl_1.42.tar.gz
apache_1.3.14.tar.gz
openssl-0.9.6a.tar.gz
```

Next, untar and install each file. From the `/usr/src/apache+ssl` directory, perform the following:

```
tar xvzf openssl-0.9.6a.tar.gz
cd openssl-0.9.6a
./configure
make
make test
make install
```

The default values are acceptable in most cases. The `make install` command will place Openssl files in the `/usr/local/ssl` directory. Openssl is now installed on the system. Next, return to the `/usr/src/apache+ssl` directory. Continue with the `apache_1.3.14` file:

```
cd ..
tar xvzf apache_1.3.14.tar.gz
```

Now copy `apache_1.3.14+ssl_1.42.tar.gz` into the `/usr/src/apache+ssl/apache_1.3.14` directory with the following command:

```
cp apache_1.3.14+ssl_1.42.tar.gz apache_1.3.14/
```

Untarring the `apache_1.3.14+ssl_1.42.tar.gz` will place the contents in the same directory as the other Apache files. To do this, use the following:

```
tar xvzf apache_1.3.14+ssl_1.42.tar.gz
```

Compile Apache with mod_ssl Support

The next step is to apply the SSL patch to Apache. Apache+SSL comes with a script to handle this step. If the script does not work on the system, the patch should be applied manually using the patch command. (For more information on using patch, consult the man pages. In addition, consult the `README.SSL` file for further information on configuration settings needed to edit `/usr/src/apache+ssl/apache_1.3.14/src/Configuration`.) Next, apply the patch using the Fixpatch script:

```
./Fixpatch
```

Apache is now ready for installation and configuration. The following actions should be performed:

```
./configure --enable-module=so
make
make install
```

By default, Apache files are placed in the `/usr/local/apache` directory. If everything worked properly, the following output should be displayed:

```
+--------------------------------------------------------+
| You now have successfully built and installed the      |
| Apache 1.3 HTTP server. To verify that Apache actually |
| works correctly you now should first check the         |
| (initially created or preserved) configuration files   |
|                                                        |
|   /usr/local/apache/conf/httpsd.conf                   |
|                                                        |
| and then you should be able to immediately fire up     |
| Apache the first time by running:                      |
|                                                        |
|   /usr/local/apache/bin/httpsdctl start                |
|                                                        |
| Thanks for using Apache.      The Apache Group         |
|                                http://www.apache.org/  |
+--------------------------------------------------------+
```

Verify That Apache Was Compiled with mod_ssl

The next step involves creating a certificate. Go into the `/usr/src/apache+ssl/apache_1.3.14/src` directory and make the certificate:

```
cd src
make certificate
```

The following output is generated (fill out the information pertinent to the system):

```
You are about to be asked to enter information that will be incorporated into your
    certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
```

```
There are quite a few fields but you can leave some blank.
For some fields there will be a default value,
If you enter ".", the field will be left blank.
Country Name (2 letter code) [GB]:
State or Province Name (full name) [Some-State]:
Locality Name (e.g., city) []:
Organization Name (e.g., company; recommended) []:
Organizational Unit Name (e.g., section) []:
server name (e.g., ssl.domain.tld; required!!!) []:
Email Address []:
```

Test the Sample Page in a Web Browser

The final step is to replace the Apache configuration file (`httpd.conf`) with the SSL file. The location of the `httpd.conf` file varies with distribution. The following shows Red Hat's default location:

```
$  cp /usr/src/apache+ssl/apache_1.3.14/SSLconf/conf/httpd.conf
/usr/local/apache/conf/.
```

The default settings work in most cases. If changes are made to `httpd.conf`, check the configuration before starting the Apache daemon, as shown here:

```
/usr/local/apache/bin/apachectl configtest
/usr/local/apache/bin/apachectl start
```

The server can be checked by bringing up a browser and connecting to http://localhost. An OpenSSL certificate should be displayed on the screen.

Questions

1. Give one way to check that the Apache+SSL installation is running correctly.
2. Can a "normal user" employ `apachectl`?
3. Why are SSL and Apache distributed separately?

Answers

1. From the server, Lynx localhost or http://localhost in Netscape Navigator, Internet Explorer, or another browser.
2. No, `apachectl` must be run as root.
3. Due to export restrictions imposed by the United States and other nations, some software cannot be distributed with certain types and levels of encryption. As a result, Apache can be distributed only without the encryption and other features offered by SSL.

Advanced Questions

1. How does Apache+SSL negotiate content between the server and client?
2. What encryption methods are available for Apache+SSL?

LAB

III

Configuring Apache to Perform Common Tasks

Purpose

The purpose of this lab is to set up an Apache server with several common Web server features. Apache is the most commonly used Web server because it is freely available and can be customized to suit almost any need for a Web server.

In this lab, you will perform the following steps:

1. Allow cgi-scripting in normal document directories outside of the existing `scriptalias cgi-bin`.
2. Add a virtual host.

For this lab, your computer should have Apache+SSL preinstalled.

Theory

When Web pages were static, interaction was one way and the server simply sent requested pages. This method was effective, but it relied on the

Web pages to be updated constantly or customized for a particular user. To offer dynamic content and input from the user, Common Gateway Interface (CGI) scripts were used to unlock the data residing on servers. CGIs are programs that run on the server and send output to the user's browser. Almost any program that can run on a server can be used as a CGI script. The most common of these are Perl scripts, JavaScript, and PHP. These programs are usually found in the `cgi-bin` subdirectory on the server. There is typically a system-wide `cgi-bin` subdirectory, but it can also exist in a user's directory on the server. Placing the `cgi-bin` in the user's directory will be one of the tasks performed in this lab.

First, open the configuration file for Apache, `httpd.conf`, in a text editor. The location of added code is not crucial, but it makes future edits easier if all the CGI directives are together. On most default systems, there is a default CGI directive similar to the following:

```
<Directory "/var/lib/apache/cgi-bin">
        AllowOverride None
        Options None
        Order allow,deny
        Allow from all
</Directory>
```

Add the following code to enable CGI scripts in a user's home directory:

```
<Directory "/home/user/public_html/cgi-bin"> - specifies the directory containing
   scripts
     AllowOverride None - specifies whether a directive .htaccess should control
     Options ExecCGI - executes all CGI scripts in the directory
     Order allow,deny sets the order of permissions in .htaccess or in following lines
     Allow from all - grants permission for all users to execute scripts
</Directory>
```

Save and restart Apache with the following:

```
apachectl restart
```

Next, a name-based virtual host will be configured. Among the many features offered by Apache is the ability to host different sites on the same machine. Small to medium-sized Web sites can typically be run off a medium-grade Pentium machine with a 10-bit Ethernet connection. A 10-bit

connection often provides more than enough bandwidth for these sites; in this case, an additional site may be run. Therefore, the next task will be to set up a virtual host.

There are three separate ways to host multiple Web sites on a single machine:

- Use multiple instances of the daemon
- Use IP-based virtual hosting
- Use name-based virtual hosting

Similar to IP-based hosting, name-based hosting allows multiple server names to point to one machine. One of the major benefits is that a number of hosts can use a single IP address, avoiding the need to obtain an additional IP address. Name-based hosting is available because of a feature available in most browsers. When a browser requests a page, the destination name is included in the header. Apache is able to use this information and direct the request to the location holding the information.

Configuring a system to use name-based hosting takes place in the `httpd.conf` file. In most distributions, the virtual host section is commented out; the easiest way to implement virtual hosting is to remove the comments. The following steps show how to add name-based hosting to a site:

1. Open the Apache configuration file (`httpd.conf`) in a text editor.
2. Add the following text to the file:

```
<VirtualHost 123.45.67.89> - specifies the IP address for name-based hosting
ServerName www.site1.com - states the requested name for the first website
ServerAdmin webmaster1@site1.com - specifies the email for the administrator
DocumentRoot /home/web/site1 - sets the directory containing the first website
ErrorLog /var/log/site1-error_log - sets the error log directory for the first
     site
</VirtualHost>
#Second host
<VirtualHost 123.45.67.89> - specifies the IP address for name-based hosting
ServerName www.site2.com - states the requested name for the second website
ServerAdmin webmaster2@site2.com - specifies the email for the administrator
DocumentRoot /home/web/site2 - set the directory containing the second website
ErrorLog /var/log/site2-error_log - sets the error log directory for the second
     site
</VirtualHost>
```

3. Make sure the specified directories have the appropriate content. Restart the `httpd` daemon to read the new configuration with the following:

```
apachectl graceful
```

Lab Exercises

To perform these exercises, you will need to locate and understand the Apache configuration files under `/usr/local/apache/`. All the other Apache information is also located here. Make sure you find the configuration files before proceeding.

To successfully debug these exercises, it will also be useful for you to locate the Apache logs, particularly the error log. Again, these are located in various places. Red Hat puts them in `/var/log/httpd` (and sym links them from `/etc/httpd/logs`). The Apache distribution (unmodified) puts them in `/usr/local/apache/logs/`.

Once you have found the appropriate Apache files and logs, follow these steps:

1. Reconfigure Apache to give each of your users (~username paths) a private `cgi-bin` directory, either under the user's Web document root or in another place in the user's home directories.
2. Test the `cgi-bin` directory by writing a small Perl script or shell script that prints `hello`.
3. Bind in a second IP address (192.168.1.250) for Apache to use as a virtual site.
4. Configure Apache to use the new IP address as an IP virtual server.

Questions

1. What is the maximum number of name-based virtual hosts that a single machine can accommodate?
2. What script is commonly used to restart the `httpd` daemon?
3. Name three common languages used to produce CGI scripts?

Answers

1. There is no maximum number. More virtual hosts, though, mean more resources consumed, which affects overall system performance.
2. `Apachectl`.
3. Perl, JavaScript, and PHP.

Advanced Questions

1. Why would you want to put the `cgi-bin` directory outside of the document root?
2. What is the danger of having multiple `cgi-bin` directories scattered throughout a server?

LAB

IV

Create a Simple CGI Script

Purpose

This lab exercise will teach you to create a simple CGI script and execute it from both a command line and a Web browser. This lab will allow you to configure Apache to accept CGI scripts in a given directory, write a simple CGI script and make it executable, give parameters to a CGI from a URL, and process view parameters in the CGI's output.

Theory

The Common Gateway Interface (CGI) can do tasks that HTML cannot accomplish alone. CGI programs run on the server side, allowing you to create forms and guestbooks, among other things. CGI scripts are written most commonly in Perl, but they can also be written in C, C++, shell scripting, Python, TCL, and other languages.

CGI scripts are made available to a browser through the use of simple links, specialized URLs containing question marks, HTML tags, or HTML+FORMs. After an HTML document containing one of these elements is submitted, a query is passed to the HTTP server, which is then

passed to the CGI script itself. The script processes the input, formats the output into HTTP codes or an HTML document, and returns the codes or documents to the HTTP server. This information is then passed along to the client application.

Lab Exercises

CGI scripts can be shell scripts. This lab exercise will allow you to create a simple CGI script and execute it.

Create a Basic CGI Script

NOTE **All tasks from here forward will be in the working directory /usr/local/apache.**

Configure Apache for `/cgi-bin/` scripting as root by adding the following lines to your `httpd.conf` file:

```
ScriptAlias /cgi-bin/ /home/httpd/cgi-bin/
<Directory "/home/httpd/cgi-bin">
AllowOverride None
Options None
Order allow,deny
Allow from all </Directory>
```

NOTE **Paths may vary based on your installation.**

Save `httpd.conf` and restart Apache.

Check the log to see Apache's status:

```
$ tail /var/log/httpd/error_log [Wed Dec 20 17:48:06 2000]
[notice] SIGHUP received.  Attempting to restart [Wed Dec 20 17:48:10 2000]
[notice] Apache/1.3.12 (Unix) mod_perl/1.21 PHP/3.0.16 configured -- resuming normal
operations
```

Once you see the `resuming normal operations` message, create the first script. Create this file as `/var/lib/apache/cgi-bin/set.sh` as follows:

```
#!/bin/sh
# set.sh -- show the environment in a CGI script.
echo Content-type: text/plain
echo
set
```

`chown` this file to the user and group under which Apache's `httpd` process runs, and mark it executable. In this example, Apache is run by user.group `nobody.nobody`:

```
$ chown nobody.nobody /var/lib/apache/cgi-bin/set.sh
$ chmod +x /var/lib/apache/cgi-bin/set.sh
```

Run `chmod` from the command line; note the output.

```
$ ./set.sh
```

If the output scrolls off your screen before you get a chance to view it, pipe (|) it into `less`:

```
$ ./set.sh | less
```

Use the space bar to scroll through the output.

Now try it from the Web browser. Open http://localhost/cgi-bin/set.sh and view the contents there. Compare the two outputs.

Give the set script a parameter; open http://localhost/cgi-bin/set.sh?test. Try it with different parameters. Check the value of the `QUERY_STRING` variable each time.

Questions

1. Why change the owner of the script to the owner of Apache?
2. Assuming you have the correct owner of the file, what is the tightest level of permission you can have on the file and still have it execute?

Answers

1. To give access to those viewing the page or script.
2. `-rwx------.`

Advanced Questions

1. `set` is a shell command. What other shell commands are useful in CGI programming? What are some potentially dangerous shell commands?
2. How can you use the `QUERY_STRING` variable for more complex shell scripts?
3. Why is the output in your Web browser different from the output obtained when you execute on the command line?
4. Why are there two `echo` commands in the bash script (i.e., `echo Content-type: text/plain echo`)?

LAB

V

Configure and Run mod_auth_mysql

Purpose

In this lab, you will edit the `httpd.conf` file to use `mod_auth_mysql` for HTTP user authentication. This will configure Apache to use MySQL as the primary username and password database. Students will then gain familiarity with administering a MySQL user database. The configuration will be tested using a Web browser.

Theory

To increase security, the Apache Web server can require authentication before processing a client's request. There are several ways in which Apache can store usernames and passwords. Although Apache can use text files to store authentication data, it is often more secure and efficient to administer this data through a MySQL database. This is done by using the `mod_auth_mysql` module to enable authentication and to specify the use of the MySQL database.

Setting Up the MySQL Database

Before Apache can use `mod_auth_mysql` to perform authentication, a database must exist to store the usernames and passwords of authorized users. The `mysqladmin` command is used to create this database. When the database has been created, it is necessary to set up the appropriate database fields properly. After the MySQL database has been constructed, MySQL must be configured to allow the Web server to access the newly created database. MySQL must then be restarted so that the changes can take place. It is also necessary to create a test user so that the configuration can be tested later.

Setting Up Apache

With the MySQL database properly set up, the next step is to configure Apache to use this database for authentication. This is done by first editing the `httpd.conf` file to use the `mod_auth_mysql` module for authentication. With the authentication scheme in place, a test directory is needed with a `.htaccess` file to protect the directory. After making changes to Apache's configuration file, the `httpd` daemon must be restarted to make the changes take effect. The authentication can then be checked by accessing the page with a Web browser.

Lab Exercises

To complete this lab, a computer with Apache+SSL with MySQL `mod_auth_mysql` already installed must be available.

Assuming you have added the path to the `mysql bin/` directory to your `PATH` environment variable, type:

```
mysqladmin create http_auth -u root -p
```

Enter MySQL's root password at the password prompt to access `mysqladmin`. Log into that database by typing:

```
mysql -u root -p http_auth
```

Enter your password at the prompt.

Now it is time to add a simple table to your database, which you can use to grab usernames and passwords. mod_auth_mysql has some default field names for this table, which we will use. Set up the table as follows:

```
create table mysql_auth (
username char(25) not null,
passwd char(25) not null,
groups char(25),
primary key (username)
);
```

As you can see, this is a very simple table. In MySQL, the char() field is designed specifically to hold characters. While it may seem unwise to store user passwords as clear text in this database, mod_auth_mysql encrypts the passwords by default by using the MySQL password() routine.

We also built the table with two fields that are labeled not null, meaning that neither the username nor the password can be null when creating new users.

For security purposes, you want to create a new username and password that mod_auth_mysql can use to access the database you have created. Because you are currently logged in as root, this will be easy. Type the following:

```
GRANT DELETE, INSERT, SELECT, UPDATE ON http_auth.* TO apache_user@localhost
IDENTIFIED BY 'kl4u137';
```

This gives the user apache_auth the ability to remove information, insert information, find information, or change information on any table within the mysql_auth database, as long as apache_auth is connecting from the local host. Before the new user can go into effect, we need to flush MySQL's privilege tables. To do this, perform the following steps:

1. If you are still in MySQL, type exit; hit return. From the command line, type the following:
   ```
   mysqladmin -u root -p reload
   ```
2. Enter your password at the prompt. This command reloads the permissions so that MySQL will recognize your new user. To test, try to enter the database as apache_user with a password of kl4u137, as shown here:
   ```
   mysql -u apache_user -p http_auth
   ```

3. Enter `kl4u137` at the prompt. Now that you are in, use the following code to add a new user, which Apache and `mod_auth_mysql` can use to validate:

```
INSERT INTO mysql_auth (username, passwd, groups) VALUES ('user_1',
password('c0ff33'), 'test')
```

 `user_1` will be the first Web-based user for us to test.

4. Notice the use of password `c0ff33` as the value that went into the `mysql_auth` `passwd` field. Type the following:

```
select * from mysql_auth;
```

 You will notice that the password column does not look like `c0ff33` at all. `password()` is the default function `mod_auth_mysql` uses when it checks passwords.

5. Open your `httpd.conf` file and add the following lines:

```
Auth_MYSQL on apache_user
Auth_MySQL_Info localhost apache_user kl4u137
Auth_MySQL_Encryption_Types MySQL
```

 `Auth_MySQL_Info` designates the standard database you are using, the username and password you use to connect to it, and the location from which you plan to connect (in this case, `localhost`).

 `Auth_MySQL_Encryption_Types` is the method by which `mod_auth_mysql` will encrypt your passwords. There are three options here:

 - Plain text
 - Crypt_DES
 - MySQL

 You can use one or all of these options, and the module will use each method to check your database. Because we inserted our user with the `password()` function in MySQL, it makes sense to choose the MySQL option here.

6. Next, select a directory you would like to protect using `mod_auth_mysql`. Create a `.htaccess` file in this directory, and place the following code in it.

```
AuthName –Super Secret Web siteÓ
AuthType Basic
require valid-user
```

7. Restart Apache and then try to view a Web page within the newly protected directory with your Web browser. When you see the popup

window asking for your username and password, enter the combination that you entered into MySQL.

Student Resources

You can find additional information at the following URLs:

MySQL authentication module for Apache at www.diegonet.com/support/mod_auth_mysql.shtml `(mod_auth_mysql v2.20)`.

Using MySQL for user authentication with the Apache server at http://bignosebird.com/notebook/mysqlauth.shtml.

Questions

1. Could you use `.htaccess` files with `mod_auth_mysql`?
2. Because `mod_auth_mysql` uses the same directives as if you just used text files, how would you check a user's group as well?
3. Could you add columns to this table to hold more user information, possibly for an online community situation?

Answers

1. Yes, but it would be slower.
2. Add `require group test`, where `test` is the name of the group you require.
3. Yes.

Advanced Questions

1. You can use `Auth_MySQL_DB [database_name]` within a given scope to point to a database different from the default. What purposes could this serve?

2. What is the purpose of using `Auth_MySQL_Encryption_Types` with one or more encryption methods? More specifically, what does this directive secure against?
3. What are the security ramifications of storing a MySQL username and password in `httpd.conf`?

LAB

VI

Apache and Tomcat

Purpose

The purpose of this lab is to configure Apache to work in tandem with Jakarta-Tomcat. Jakarta-Tomcat will serve dynamic content, and Apache will serve static content.

Theory

One of the strengths of Apache is that it can be customized to meet almost any need. Traditionally, Apache utilized programming languages, such as C and Perl, to offer dynamic content for Web pages. Java servlets offer an alternative to CGI scripts. In addition to providing dynamic content, Java code can be run on any platform using a Java Virtual Machine, with little or no modification to the source code. In other words, the code can be ported to other platforms easily.

Before beginning the Jakarta-Tomcat installation, the system must have either Java Runtime Environment (JRE) or Java Development Kit (JDK) installed. This software must be installed properly along with the proper paths and classpaths before Tomcat will work properly.

While Jakarta-Tomcat can be utilized as a stand-alone Web server, using it along with Apache has its advantages. They are as follows:

- Apache hosts static Web pages faster than Tomcat.
- Apache has more configuration options than Tomcat.
- Tomcat, along with Apache, can host CGI scripts as easily as Java servlets.

Lab Exercises

1. From the Web site http://jakarta.apache.org download the latest stable binary version of Jakarta-Tomcat; as of this writing, the latest version is 3.2.1. For simplicity, put the Jakarta-Tomcat tarball in the same directory as `jdk` (e.g., `/opt`):

```
cd /opt
tar xvzf jakarta-tomcat-3.2.1.tar.gz
```

2. The next step involves editing `/etc/profile` to include the following environment variables:

```
export CLASSPATH=/opt/jakarta-tomcat-3.2.1/src.jar:/JAVA-DIR/jakarta-
     tomcat-3.2.1/ servlet.jar:
export TOMCAT_HOME=/opt/jakarta-tomcat-3.2.1
```

3. Next, download the `mod_jserv.so` from http://jakarta.apache.org/builds/jakarta-tomcat/release/ v.3.2.1/bin/linux/i386. Place this file in the `/JDK/jakarta-tomcat-3.2.1` directory.
4. Copy the `tomcat-apache.conf` file (or `tomcat.conf`) to `tomcat-jserv.conf`. Edit the Apache configuration file, `httpd.conf`, to include the following line:

```
Include /opt/jakarta-tomcat-3.2.1/tomcat-jserv.conf
```

5. Make sure the `mod_jserv.so` file is referenced correctly in `tomcat-jserv.conf`. The first line of `tomcat-jserv.conf` should include the following:

```
LoadModule jserv_module /opt/jakarta-tomcat-3.2.1/mod_jserv.so
```

6. Restart Apache using the init scripts or the apachectl script (e.g., `/etc/rd.d/init.d/httpd restart` or `/etc/httpd/apache restart`).

7. Start Tomcat using the following:

```
/opt/jakarta-tomcat-3.2.1/bin/startup.sh
```

8. To test the new setup, point your browser to http://localhost/examples/servlets/.
9. Click on the links to execute the various examples of servlets.

Questions

1. Why would one use Tomcat in conjunction with Apache?
2. What technologies, other than Java servlets, provide dynamic content?

Answers

1. Tomcat, along with Apache, can host CGI scripts as easily as Java servlets.
2. CGI scripts written in Perl, C, or shell scripts.

Advanced Questions

1. Compare and contrast Java servlets and CGI scripts.
2. Write a servlet to display user input.

LAB

VII

Configuration of a Proxy

Purpose

The purpose of this lab is to teach you how to configure a caching HTTP proxy. A proxy can perform some of a firewall's functions, but typically it is used to allow users to access Web pages from behind a firewall or to speed up performance of an Internet connection.

Theory

A proxy server is a machine that processes requests for data from one system, then passes that request on to the proper server. Proxies are often used to process HTTP requests through a firewall. If the proxy also saves requested data, it becomes a caching proxy. Because they store information locally and do not need to access the primary server, caching proxies allow more efficient use of the network. Apache's functionality may be extended to include proxy features by using the module `mod_proxy`.

Installing mod_proxy

The first step to setting up Apache as a proxy server is to install `mod_proxy`. Apache can be updated with `mod_proxy` in two ways. First, Apache can be recompiled to include `mod_proxy` in the core distribution. Second, if Apache has Dynamic Shared Object (DSO) support, `mod_proxy` can be inserted as a module at run time. For this lab, `mod_proxy` will be loaded as a module at run time. Therefore, Apache must have `mod_so` compiled, which will provide loadable module support.

Configuring httpd

After `mod_proxy` has been integrated into Apache, `httpd.conf` must be changed to take advantage of the proxying features. This is done by adding directives recognized by `mod_proxy` to `httpd.conf`. Table VII.1 lists some of the most common directives supported by `mod_proxy`.

A more detailed list of directives for `mod_proxy` can be found on the Apache Web site: http://httpd.apache.org/doc/mod/mod_proxy.html.

After making changes to `httpd.conf`, the Web server must be restarted for the configuration changes to take place.

Table VII.1 mod_proxy Directives

DIRECTIVE	DESCRIPTION
`ProxyRequests`	Determines if Apache will function as a proxy
`ProxyRemote`	Specifies a proxy for Apache to use
`ProxyPass`	Converts requests for local data to a proxy request
`ProxyBlock`	Blocks proxy connections to specified sites
`NoProxy`	Instructs Apache to serve requests directly, instead of proxying
`ProxyDomain`	Appends the domain name within intranets
`CacheRoot`	Sets the directory to write cache files to
`CacheSize`	Sets the size of the cache in kilobytes
`NoCache`	Specifies locations not to cache from

Configuring the Client

Once the proxy server has been properly set up, any client wishing to use the proxy server must be configured to use the server. This is done by making changes to the client's Web browser settings. For the purposes of this lab, it is assumed that the client is running Netscape Navigator for HTTP connections. After the settings have been changed, the client should be able to make HTTP connections through the newly configured proxy.

Lab Exercises

The following exercises will take you through the steps of installing `mod_proxy`, configuring `httpd`, and configuring the client.

Installing mod_proxy

The mod_proxy module is required for Apache to run on a proxy server. The following steps show you how to install mod_proxy.

1. Make sure that Apache has loadable module support compiled in. From the command prompt, type:

   ```
   httpd -l
   ```

2. This will list all currently loaded modules. If `mod_so` is not listed, Apache must be recompiled to include it. If `mod_proxy` was listed, there is no need to complete the rest of this section. Skip to the next section.
3. Compile the `mod_proxy` module according to the instructions included with the distribution of Apache. `mod_proxy` is part of the base Apache distribution and is included with the Apache source.
4. To have `mod_proxy` loaded every time Apache is started, use the `LoadModule` directive. First, open `httpd.conf` with a text editor, and add the following in the Global Environment Section:

   ```
   LoadModule mod_proxy path/to/mod_proxy.so
   ```

Configuring httpd

This section shows you how to enable the proxy server.

1. Locate the proxy directives in the `httpd.conf` file that came with Apache. Uncomment the lines that configure the proxy, and add any

site-specific configurations. In order for Apache to proxy, the directive `ProxyRequests On` must be present before the other `mod_proxy` directives. `ProxyPass` is the only directive that does not require `ProxyRequests`.

2. Implement the caching abilities of `mod_proxy` by using the directives `CacheRoot` and `CacheSize`. The value of `CacheRoot` must be set before Apache will cache HTTP data:

```
CacheRoot "/var/cache/http/"
```

 Also, make sure that `httpd` has write permissions for this directory.

3. Restart `httpd`.

Configuring the Client

Configure the user-end client (Netscape) to use the proxy server, using the following steps.

1. On the client machine, open Netscape.
2. From `Edit>Preferences`, select `Advanced>Proxies`.
3. Click `Manual proxy configuration`, and then click `view`.
4. In the HTTP proxy field, type the host name and port number that the proxy uses.

Questions

1. If a company runs an intranet and uses the intranet server as the proxying firewall, which directive should be used so the server will not proxy intranet requests?
2. How can modules be integrated into Apache?
3. If the directives `ProxyRequests Off` and `ProxyPass` are both present in `httpd.conf`, what will occur?

Answers

1. The `NoProxy` directive will make a proxy server handle specified requests, instead of proxying the request.
2. Apache can be recompiled to include the module, or if Apache has DSO support, the module can be inserted with the `LoadModule` directive.
3. The `ProxyPass` directive will function properly because it does not require `ProxyRequests`.

Advanced Questions

1. How can Apache be set up to proxy FTP requests?
2. How can `mod_proxy` directives be included only on the condition that `mod_proxy` is loaded?
3. How can Lynx be configured to use the Apache proxy?

LAB

VIII

URL Rewriting

Purpose

The lab will teach students the basics of URL rewriting, using Apache's URL rewriting engine. Students will attempt two tasks involving `mod_rewrite`: rewriting a URL to point to a different Web server and rewriting a URL to a different file based on browser type.

Theory

Apache has a powerful tool to redirect certain URLs based on pattern matching. The method that Apache uses to redirect URLs is to rewrite the URL sent to the server with the URL to which the client should be redirected. This is useful because it provides backward compatibility for outdated URLs. To be able to rewrite URLs, the server must have `mod_rewrite` installed and have appropriate patterns for the `RewriteRule` directive to match.

`mod_rewrite` comes with several directives that can be used to rewrite URLs. Several important `mod_rewrite` directives are listed in Table VIII.1.

Table VIII.1 mod_rewrite Directives

DIRECTIVE	DESCRIPTION
`RewriteEngine`	Enables URL rewriting
`RewriteLog`	Sets the rewriting log file
`RewriteLogLevel`	Determines the logging level
`RewriteCond`	Provides conditionality of rewriting
`RewriteRule`	Specifies actual rewriting rules

The `RewriteEngine` directive is the most important directive and must be set On in order to be able to make use of URL rewriting.

`RewriteRule` is the most common directive and is used to actually change a URL. Every URL that should be changed must start with the `RewriteRule` directive, which takes the form:

```
RewriteRule pattern substitution [flags]
```

`pattern` is a regular expression for Apache to match, and substitution is the string that should be substituted into a matching URL. `flags` are optional modifiers that alter the directive. `mod_rewrite` recognizes several wildcards when rewriting a URL, and several flags can be appended to a `RewriteRule` directive. `pattern` must start with (`^`) and end with (`$`). `RewriteRule` also performs wildcard expansion and can backreference selected text. An example of backreferencing is this:

```
RewriteRule ^/(important\.docs) /secret/$1
```

In this example, the `$1` backreferences `important\.docs` (the backslash is used to escape the period). Several backreferences can be used, by changing `$1` to `$N`, where `N` represents the Nth backreference. Also supported are the wildcards (`.`), (`*`), and (`+`). The period matches only one character, the asterisk matches zero or more characters, and the plus matches one or more characters.

Some of the flags that can be appended to a `RewriteRule` directive are listed in Table VIII.2.

Table VIII.2 RewriteRule Flags

FLAG	DESCRIPTION
F	Return an http 403, forbidden, error
G	Return an http 410, gone, error
P	Perform an internal proxy request to substitution via the ProxyPass directive
R	Perform an external redirect
NC	Pattern is matched without respect to case
C	Chain the current rule with the next rule
L	Do not apply any more rewriting rules

`RewriteCond` is also commonly used so that a condition may be tested before executing a `RewriteRule` directive, much like an if statement when programming. The syntax for `RewriteCond` is:

```
RewriteCond testpattern condpattern [flags]
```

where `testpattern` is checked against `condpattern` and flags are optional modifiers. This can be remembered by the statement "if `testpattern` matches `condpattern`, execute the following code." The `RewriteCond` syntax includes the use of server variables, specified as `%{variable}`, and backreferences to text grouped by parentheses, like `RewriteRule`.

Some of the server variables supported by `RewriteCond` are these:

```
REMOTE_ADDR   REMOTE_HOST   REMOTE_USER   DOCUMENT_ROOT
SERVER_NAME   SERVER_ADDR   SERVER_PORT   SERVER_PROTOCOL
TIME_YEAR  TIME_MON  TIME_DAY  TIME_HOUR  TIME_MIN  TIME_SEC
HTTP_USER_AGENT  HTTP_REFERER  HTTP_HOST  HTTP_ACCEPT
```

A complete list can be found in the documentation for `RewriteCond` on the Apache Web site at http://httpd.apache.org/docs/mod/mod_rewrite.html.

A basic example of RewriteCond is this:

```
RewriteCond %{REMOTE_HOST} ^theserver.com
RewriteRule ^/$ /theserverpage.html
```

This would rewrite all URLs to DocumentRoot/theserverspage.html only if the variable REMOTE_HOST matches theserver.com.

Additionally, RewriteCond supports two optional flags: [OR] and [NC]. [OR] is used to test multiple conditions, while [NC] is used to disable case sensitivity for both testpattern and condpattern. For example:

```
RewriteCond %{REMOTE_HOST} 192.168.3.14 [OR]
RewriteCond %{REMOTE_HOST} ^thatserver.com [NC]
RewriteRule ^/$ /theserverpage.html
```

will match the hosts 192.168.3.14, thatserver.com, or That-SERver.com.

For additional help constructing regular expressions, consult the regex man page.

Lab Exercises

For this lab, it is assumed that the user has Apache running, with mod_rewrite installed. mod_rewrite will be used to rewrite URLs for two different cases: redirecting from one server to a new server and redirecting a user to a browser-specific page, based on the Web browser being used to access the server.

Using the RewriteRule directive, add a line to httpd.conf to redirect a user from http://oldserver/~theuser/somepath to http://newserver/~theuser/somepath.

Using the RewriteCond conditional statement, search for a user's client type and then, with RewriteRule, redirect the user to a client-specific page.

1. Open httpd.conf in a text editor.
2. Redirect the home directories from one location to another using mod_rewrite.

Use `mod_rewrite` to redirect all URLs with `/~theuser/somepath/` to http://newserver/~theuser/somepath/ with the following:

```
RewriteEngine on
RewriteRule  ^/~(.+)$  http://newserver/~$1  [R,L]
```

3. Using `RewriteCond` redirect a user based on the client.

```
RewriteCond %{HTTP_USER_AGENT}  ^Mozilla/3.*
RewriteRule ^foo.html$         foo.NS.html       [L]
RewriteCond %{HTTP_USER_AGENT}  ^Lynx/.*          [OR]
RewriteCond %{HTTP_USER_AGENT}  ^Mozilla/[12].*
RewriteRule ^foo.html$         foo.20.html          [L] RewriteRule ^foo_html$
foo.32.html          [L]
```

4. Save the changes to `httpd.conf`.
5. Restart `httpd`.

Questions

1. How can a `mod_rewrite` be used to change a URL only if a certain condition is met?
2. Before any rewriting rules will function properly, what directive must be present in `httpd.conf`?

Answers

1. The `RewriteCond` directive must be specified, along with a matching statement, in order to provide a matching condition.
2. `RewriteEngine` On must be present before `mod_rewrite` will work.

Advanced Questions

1. Are there any special issues if the redirected home directories are NFS mounted?

2. Is it possible to redirect a user through a proxy server?
3. How can a `testpattern` be compared to a condpattern lexically in a `RewriteCond` directive?

LAB

IX

Create a Custom Log for Apache

Purpose

The purpose of this lab is to create and implement a custom log file that tracks remote IP-addresses, lists the `REFERRER` environmental variable, the time the request occurred (common log format), and the time it takes to serve the request. Optionally, you can run `logresolve` to resolve the IP addresses, as well as view the log file using a popular log-viewing program.

Theory

By default, Apache comes with basic logging capabilities. However, the logging capabilities of Apache can be greatly expanded by using the `mod_log_config` module. This module allows the administrator to set custom log definitions through the `httpd.conf` file. `mod_log_config` contains several different directives used to configure the log file, but the main directive of interest is `LogFormat`, which is used to specify what data is to be placed in the log and in what format. Other `mod_log_config` directives are listed in Table IX.1.

Table IX.1 mod_log_config Directives

DIRECTIVE	DESCRIPTION
`CookieLog`	Sets the file name for logging cookies
`CustomLog`	Sets the file name of the custom log
`TransferLog`	Similar to custom log, with some restrictions

The basic syntax for the `LogFormat` directive is as follows:

```
LogFormat formatoptions name
```

where `formatoptions` specifies the format to write and `name` is an identifier used by `CustomLog` Directives.

The most useful characteristic of `mod_log_config` is its ability to configure the data contained in a log file. This is done by providing a list of options to the `LogFormat` directive. A list of some of the options is provided in Table IX.2.

Table IX.2 Options for mod_log_config

OPTION	DESCRIPTION
`%a`	Remote IP address
`%A`	Local IP address
`%B`	Bytes sent, not including headers
`%D`	Time taken in microseconds
`%h`	Remote host
`%H`	Request protocol
`%P`	PID of serving process
`%r`	First line of request
`%t`	Time in CLF
`%u`	Remote user
`%U`	URL path requested

The codes shown in Table IX.2 can be included on a conditional basis by placing an HTTP code between the percent sign (%) and the letter. For example, `%404U` will put the requested URL in the log file if an HTTP 404 error is encountered. By putting an exclamation point (!) before the HTTP code, the negative condition will cause logging. Text may also be put in a log file entry by placing it within braces (`{}`) and inserting it between the percent sign and the letter.

Once a format has been specified in `httpd.conf`, it must be implemented via either `CustomLog` or `TransferLog` directives. `CustomLog` is more commonly used because it is more versatile. The typical format for a `CustomLog` directive is this:

```
CustomLog file name
```

`file` specifies the file to which to log, and `name` is the name used in the `LogFormat` directive.

Lab Exercises

NOTE **This lab assumes that the user has Apache with mod_log_config installed.**

1. Open `httpd.conf` in a text editor.
2. In the main section of the file, add a `LogFormat` directive that lists remote IP addresses, lists the `REFERRER` environmental variable, the time the request occurred (common log format), and the time it takes to serve the request.
3. Create a name to reference this configuration.
4. Add a `CustomLog` line that logs data to `/var/log/access_log.test` using the format created in Step 2.
5. Restart Apache.
6. Make a request to the Web server from another machine.
7. Open `/var/log/access_log.test` in a text editor to test the configuration.

Questions

1. What is the default value for a log entry?
2. What format will `TransferLog` use if it is specified instead of `CustomLog`?

Answers

1. LogFormat "%h %l %u %t '%r' %>s %b".
2. TransferLog will use the last specified LogFormat line.

Advanced Questions

1. How would you limit your log to only certain types of server/client interactions (i.e., a 403 error)?
2. Can `CustomLog` be used to make `syslogd` handle custom Apache logging? If yes, how?

LAB

X

Benchmark Your Server

Purpose

The purpose of this lab is to use several built-in monitoring features to monitor Apache's performance. Various tools, such as Apache Bench (`ab`), `mod_status`, and `mod_info`, are used to establish a baseline for performance.

Theory

Apache is equipped with a tool for benchmarking its performance. Apache Bench (`ab`), the HTTP server benchmarking tool, does this by giving an indication of how many requests per second the server can handle, along with other statistics. These requests simulate real-world activity and aid in diagnosing potential bottlenecks in server performance.

The most common options used with `ab` are as follows:

`-n` Specifies the number of requests. Default is a potential request.

`-t` Specifies the number of seconds ab should spend performing the benchmark. By default there is no time limit.

`-c` Lists the number of simultaneous requests to perform. The default is one or no concurrency.

`-w` Prints results in HTML tables.

The benchmark results may lead to changes to the server configuration file. Apache provides a means of starting, stopping, and obtaining information about the use of `apachectl`. The Apache HTTP server control interface (`apachectl`) provides a convenient means of controlling the `httpd` daemon. As the heart of Apache, the `httpd` daemon handles all Web page requests. Each time the configuration file for the `httpd` daemon, `httpd.conf`, is changed the daemon must be restarted to read to new settings. The `httpd` daemon can be stopped and restarted manually, but `apachectl` provides an easier means of accomplishing this, along with providing other information. Table X.1 shows a list of options available for the `apachectl` script.

Table X.1 Options for apachectl

OPTION	DESCRIPTION
`Start`	Starts the Apache daemon.
`stop`	Stops the Apache daemon.
`fullstatus`	Shows the full server status. (This feature requires the mod_status module and the Lynx browser.)
`restart`	Sends a SIGHUP signal to the daemon, which restarts the daemon or starts the daemon if it is not running.
`graceful`	Once given, the daemon will not accept any new requests; it will finish all pending requests before it restarts. The method is less.
`configtest`	This option tests the configuration file for errors. This is a syntax check, not an efficiency check.
`status`	Gives an abbreviated status screen (requires mod_status and Lynx.)
`help`	Lists all the options available for apachectl.

Lab Exercises

1. Be sure the following files exist: /www/users and /www/groups.
2. Run `htpassword -c /www/users <username>`.
3. Add the `<username>` to the test group. For example, edit `/www/groups` and add the line `test: <username>`.
4. Add the following section to the bottom of the configuration file:

```
ExtendedStatus On
<Location /status>
    SetHandler server-status
    AllowOverride Authconfig
    AuthType Basic
    AuthName "Private Performance Stuff"
    AuthUserFile /www/users
    AuthGroupFile /www/groups
    require group test
</Location>

<Location /server-info>
    SetHandler server-info
    AllowOverride Authconfig
    AuthType Basic
    AuthName "Server information"
    AuthUserFile /www/users
    AuthGroupFile /www/groups
    require group test
</Location>
```

5. Save the file and close it.
6. Restart `httpd` so that it reads the edited configuration file:

```
# apachectl restart
```

7. Start Lynx so that it will access the status information on the server:

```
# lynx localhost/status
```

 The page will start to load but then complain about retrying because of authorization requirements. This will leave the user with a prompt to enter a username:

```
Username for 'Private Performance Stuff' at server \ 'localhost':
```

 At the prompt, enter your username and password.

Test the `info` page by pointing the browser to http://localhost/server-info.

Enter the username and password again.

8. Test your server's ability to take a load by running ab.

 Type the following in the command line:

   ```
   ab -n1000 -c100 http://localhost/server-info
   ```

 `ab` will run Apache through its paces in a few seconds. When it is done, the user will see something that is similar to the following:

   ```
   Server Software:        Apache+SSL/1.3.14
   Server Hostname:        localhost
   Server Port:            80

   Document Path:          /server-info
   Document Length:        48436 bytes

   Concurrency Level:      100
   Time taken for tests:   31.410 seconds
   Complete requests:      1000
   Failed requests:        0
   Total transferred:      49806228 bytes
   HTML transferred:       49584888 bytes
   Requests per second:    31.84
   Transfer rate:          1585.68 kb/s received

   Connection Times (ms)
                 min   avg   max
   Connect:        0   294   906
   Processing:   591  2672  4797
   Total:        591  2966  5703
   ```

9. Open `server-status` in your favorite Web browser again. Run `ab` from the command line just as before. While it is running, hit reload on your server-status page. If you previously ran `ab` from within X Window, drop out of X and run the test again, saving it in a text file like so:

   ```
   ab -n1000 -c100 http://localhost/server-info > Xless_ab.txt
   ```

 Go back into X and run the same test, saving it in a text file called `Xtest_ab.txt`.

 If a partner is available, check out his or her performance. Choose http://partner's_ip_address/server-info with `ab`.

Questions

1. Which `apachectl` option will deny all new requests and finish all pending requests before restarting?
2. What is the default requests value for `ab`?
3. What is the common configuration file for the Web server daemon?

Answers

1. apachectl graceful.
2. 1.
3. httpd.conf.

Advanced Questions

1. What does it mean if several requests fail?
2. Outside of adding more memory, how can a Web server's performance be improved?
3. What is the number of virtual hosts a Web server can handle?



Practice Questions and Answers

Practice Questions

1. Which files contain a majority of the configurations for Apache?
 A. `apache.conf`
 B. `httpd.conf`
 C. `http.conf`
 D. `cour`

2. Which of the following best defines scope for Apache?
 A. Scope is the area of effect for a given configuration.
 B. Scope relates to the number of users who can access a site.
 C. Scope is a program for monitoring Apache performance.
 D. Scope is a configuration file that targets specific content to specific IP ranges.

3. Which of the following best defines an Apache module?
 A. A small chunk of data that is served to the Web via Apache.
 B. A type of Apache server used in clusters.
 C. A piece of code, separate from Apache, used to extend Apache's functionality.
 D. A configuration file separate from the primary file.

4. How is DNS look-up used in conjunction with an Apache server?
 A. It is the process of resolving a name to its related IP address.
 B. It is a way of automatically searching the Internet through a search engine.
 C. It is a denial-of-service attack.
 D. It is a way of finding the current Apache version.

5. What is the role of a directive in Apache?
 A. It offers guidelines on how to configure Apache.
 B. It sets Internet standards, much like an RFC.
 C. It gives a request for information by a browser.
 D. It is an order given to Apache from within Apache's configuration files.

6. Which of the following will enable the module `info` during the Apache build?
 A. `make apache module=info modules_install`
 B. `/configure --enable-module=info`
 C. `make httpd module=info modules_install`
 D. `/configure --module=info`

7. In addition to installing at compile time, modules may be enabled in which of the following files?
 A. `srm.conf`
 B. `access.conf`
 C. `httpd.conf`
 D. `apache.conf`

8. Which of the following is the correct format for adding a mod_log module to the Apache configuration file?
 A. `AddModule modules/mod_log.config.o`
 B. `Module -enable=mod_log.config.o`
 C. `Module -add=mod_log.config.o`
 D. `Add module=mod_log.config.o`

9. After reconfiguring Apache, the daemon fails to restart. Which of the following files should be checked when diagnosing the problem? (Select two.)
 A. The Apache error log
 B. `/proc/apache/problems`
 C. Try running Apache under `strace`
 D. `/etc/lilo.conf`

10. How is a secure connection opened with an Apache Web server?
 A. Use https:// in the URL for the server.
 B. Enable Java in your browser.
 C. Hold down the Shift key while loading the page.
 D. Add ~secure to the end of the URL.

11. What is the purpose of `apachectl`?
 A. It allows Apache to run the CTL scripting language.
 B. It controls which users have access to Apache on a system.
 C. It allows complete configuration of Apache on remote systems through a Web-based utility.
 D. It is a script used to start, stop, or restart Apache.

12. In addition to `httpd.conf`, which of the following files may be used to configure Apache? (Select two.)
 A. `srm.conf`
 B. `yp.conf`
 C. `access.conf`
 D. `ypserv.conf`

13. Which of the following is *not* a way to run multiple Web sites off one Apache server?
 A. Name-based hosting
 B. Net-based hosting
 C. One instance of Apache for each site
 D. IP-based hosting

14. Which of the following options will configure Apache to conform to Slackware's file layout?
 A. `--with-Slackware`
 B. `--structure=Slackware`
 C. `--with-layout=Slackware`
 D. `--layout Slackware`

15. The ScriptAlias directive allows for renaming CGI scripts.
 A. True
 B. False

16. When attempting to access a Web document, a user is presented with a "Forbidden" error message. This message corresponds to which of the following errors?
 A. 200
 B. 202
 C. 301
 D. 403

17. Apache provides a means of ensuring that the program was compiled correctly. Choose the two commands that do so.
 A. `make check`
 B. `make compile check`
 C. `make --valid`
 D. `make test`

18. Which of the following is a valid means to manually patch the SSL package to the Apache source?
 A. `make patch = SSLpatch`
 B. `compile_patch SSLpatch`
 C. `patch -pl < SSLpatch`
 D. `patch -compile SSLpatch`

19. What content type is added to a CGI script to make sure the Web browser can execute the script?
 A. `<screen>Content-Type: text/html</screen>`
 B. `<screen>Content-Type: ascii/html</screen>`
 C. `<screen>Content-Type: ansi/html</screen>`
 D. `<screen>Content-Type: cgi/html</screen>`

20. Which of the following are common CGI script types?
 A. jpg, gif, png, pdf
 B. Perl, shell, C++
 C. PHP
 D. None of the above

21. Which of the following is *not* true about `.htaccess` files?
 A. They are a burden on the system's performance.
 B. They are really convenient.
 C. They are a potential security risk.
 D. They are inoperable without `.htpasswd` files.

22. In addition to password authentication, which of the following files offers a means for Apache authentication?
 A. The `srmn.conf` file
 B. The `.htaccess` file
 C. `/usr/bin/access`
 D. `/usr/bin/userconf`

23. Which of the following sets the default access state for an Apache directive to read the Deny directive first, then the Allow directive?
 A. `Directive Deny, Allow`
 B. `set Order = Deny, Allow`
 C. `Directive Set= Deny, Allow`
 D. `Order Deny, Allow`

24. Which three dimensions of every file does Apache use in content negotiation?
 A. Byte size, language, and protocol
 B. Type, content age, and language
 C. Content age, content decoding, and content attributes
 D. Type, encoding, and language

25. Must all of Apache's functionality be configured during compilation?
 A. Yes, once `httpd` is compiled, it must be recompiled to add functionality.
 B. No, run-time modules can be added using `apxs`.
 C. No, run-time modules are loaded using `inetd`.
 D. Yes, but Java-based modules can dynamically recompile at run time.

26. What is the advantage of using modules such as `mod_proxy` instead of compiling into Apache?
 A. The administrator can add and remove them as needed.
 B. The modules can be recompiled separately and updated individually.
 C. Modules reduce the size of the Apache binary, so Apache consumes fewer resources.
 D. The modules reduce the complexity of configuring Apache.

27. Which of the following changes to the client/browsers are required for the use of a proxy server?
 A. Setting the redirect to route things through InterNIC:/<url>.
 B. Setting the proxy address and/or port number on the browser.
 C. Setting the proxy port to 8080.
 D. Using the prefix proxy://www.<site>.com.

28. What is the major difference between SSI and XSSI?
 A. XSSI is an XML wrapper.
 B. XSSI is platform independent.
 C. XSSI is an XML renderer.
 D. XSSI can use conditional directives.

29. Which of the following are the tags that enclose an SSI directive?
 A. `<!--# SSI info -->`
 B. `<!--# SSI directive -->`
 C. <? SSI info ?>
 D. `<?--# SSI directive --?>`

30. Why is the exclamation point (!) employed in matching conditional patterns that use `mod_rewrite`?
 A. It shows that the pattern takes precedence.
 B. It looks for the opposite of the conditional statement to be true.
 C. It comments out that particular statement.
 D. It renices the `httpd` process handling the statement.

31. At what phase does Apache parse Server-Side Includes?
 A. Phase 3
 B. Phase 4
 C. Phase 5
 D. Phase 6

32. At what phase does Apache make sure an authenticated client has access to the resource it requests?
 A. Phase 3
 B. Phase 4
 C. Phase 5
 D. Phase 6

33. Which of the following phases activates all handlers?
 A. Phase 3
 B. Phase 4
 C. Phase 5
 D. Phase 6

34. Which of the following represents the best means for an administrator to connect to and perform operations on an Apache server?
 A. ssh
 B. Telnet
 C. FTP
 D. Webmail

35. In addition to other user authentication methods, which of the following is a good location for passwords for administrators, Webmasters, and other individuals who need access to Apache server configuration files?
 A. `/etc/passwd`
 B. `/etc/shadow`
 C. `/etc/access.conf`
 D. `/etc/xpasswd`

36. What is the environmental variable describing the client accessing the Web server?
 A. `"%{User-agent}i"`
 B. `"%{User-browser}i"`
 C. `"%{Browser-agent}i"`
 D. `"%{User}i"`

37. Apache does not use the CLF format.
 A. True
 B. False

38. Which of the following would be best to use in conjunction with the `Allow` and `Deny` directives?
 A. IP address
 B. Host name
 C. Domain name
 D. The `.htaccess` file

39. What are the ramifications of using a `.htaccess` file?
 A. It increases performance because Apache has to look at only one file.
 B. It decreases performance because httpd has to search the whole tree, if enabled.
 C. It is deprecated, so it should not be used.
 D. It does not affect performance at all.

40. Which of the following best describes the effects of the `apachectl fullstatus` command?
 A. It dumps a complete status page with the use of Lynx and `mod_status`.
 B. It dumps a complete status page, using only `mod_status`.
 C. It dumps a list of current connections.
 D. It shows the same information as status, but it records the date in a log file.

41. How can you tell if Apache is installed? (Select all that apply.)
 A. Look for the `/var/apache.inst` file.
 B. Point your browser to http://127.0.0.1
 C. Use the `rpm -qa | grep apache` command.
 D. Use the `find / -name httpd` command.

42. How did Apache get its name? (Select all that apply.)
 A. Its creator was Native American, of the Apache tribe.
 B. It was a series of patches to the original `httpd` daemon, hence "a patchy server."
 C. It is an acronym for Archive Protocol And CacHing Engine.
 D. The creator liked the sound of the word.

43. What is a header?
 A. A term used to describe when the Web server crashes.
 B. The first few lines of the Apache configuration file.
 C. A part of the request/response that contains information that tells the client or the server what kind of information it is receiving.
 D. A file that keeps track of available Apache modules.

44. Which of the following commands will verify an installed RPM package file?
 A. `rpm -Vp foo.rpm`
 B. `rpm -i foo.rpm`
 C. `rpm -q foo.rpm`
 D. `rpm foo.rpm`

45. Which of the following are common package formats? (Select two.)
 A. RPM
 B. DEB
 C. NROFF
 D. MOD

46. Which of the following are valid ways to test an installation of Apache on a local host? (Select two.)
 A. `telnet localhost:80`, then type `GET /index.html`
 B. `netscape --testapache`
 C. `lynx localhost:80`
 D. `apachectl localtest`
 E. `httpd test`

47. Which of the following is true concerning the Secure Sockets Layer (SSL)? (Select two.)
 A. It allows digital signatures.
 B. It was created by the Apache community.
 C. It exists on the data link layer of the TCP networking model.
 D. It requires encryption.

48. Which of the following is true concerning Apache and SSL? (Select two.)
 A. SSL and Apache are both stand-alone Web servers.
 B. SSL allows Apache to handle the HTTPS protocol.
 C. Installing SSL with Apache requires the Apache+SSL patch.
 D. SSL allows for remote system administration of Apache.

49. What does SSL use to secure a connection?
 A. A PGP script
 B. A bit stream
 C. A cookie
 D. A session key

50. Give the `./configure` argument required to obtain the GNU layout.
 A. `./gnuconfigure`
 B. `./configure --gnu`
 C. `./configure --with-layout=gnu`
 D. `./configure --gnulayout`

51. Which of the following are valid ways to start Apache? (Select two.)
 A. `/etc/rc.d/init.d/httpd start`
 B. `apache --start`
 C. `apachectl start`
 D. `apache start`

52. Which of the following are valid ways to stop Apache? (Select two.)
 A. `apachectl stop`
 B. `apache --stop`
 C. `apache stop`
 D. `/etc/rc.d/init.d/httpd stop`

53. What does a `NameVirtualHost` configuration do?
 A. It makes your main Web site redirect users to an internal page.
 B. It configures your site so that it displays differently to machines browsing from your internal network.
 C. It specifies an IP address that should be used as a target for name-based virtual hosts.
 D. It redirects incoming requests to a second instance of `httpd`.

54. Which of the following best describes PHP?
 A. It is an encryption cipher.
 B. It is a query language for databases.
 C. It is a compiled programming language.
 D. It is a scripting language.

55. PHP can *not* be used in which situation?
 A. In conjunction with CGI
 B. As a scripting language
 C. As a dynamic shared object module for Apache
 D. As a major mode of Emacs

56. Which of the following is true concerning PHP and HTML? (Choose the best answer.)
 A. HTML is used for the Unix environment while PHP is mainly used for MacOS and Win32 environments.
 B. PHP is essentially embedded HTML used for dynamic content.
 C. PHP is used for static Web pages, while HTML is used for dynamic content.
 D. PHP is a compiled language, while HTML is interpreted.

57. What will the following code do?

```
<html><head><title>Hello</title></head>
<body>
<?php echo "Hello Class<p>"; ?>
</body></html>
```

 A. Display `"Hello"` on the page itself and display `"Hello Class"` on stdout.
 B. Display `"Hello Hello Class"` on the page itself.
 C. Make the title of the page `"Hello"` and display `"Hello Class"` on the page itself.
 D. Nothing; there is a syntax error in line 3.

58. What are the start and end tags for a block of code in PHP?
 A. `<?` is a start tag, and `?>` is an end tag.
 B. `<` is a start tag, and `>` is an end tag.
 C. `<!--` is a start tag, and `->` is an end tag.
 D. `[` is a start tag, and `]` is an end tag.

59. How should you separate lines of code in a PHP block?
 A. End the line with a slash (/).
 B. End the line with a dash (-).
 C. End the line with a semicolon (;).
 D. End the line with a colon (:).

60. Which of the following best describes CGI?
 A. CGI is a method for allowing the Web server to execute a program, triggered by a Web page (or content). It allows for a more dynamic creation of content.
 B. CGI scripts make delivering graphics over the Web easier.
 C. A CGI script will create files with the `.cgi` extension, which are viewable as image files similar to `.jpg`, `.gif`, or `.png`.
 D. CGI scripts implement the Common Gateway Interface e-commerce standard.

61. Which of the following best describes the difference between CGI scripts and PHP documents?
 A. PHP documents are a set of embedded HTML tags; CGI scripts are stand-alone, compiled binary programs.
 B. CGI scripts are a set of embedded HTML tags; PHP documents are stand-alone, compiled binary programs.
 C. CGI scripts have to be recognized as executable by the operating system; PHP documents do not.
 D. PHP documents have to be recognized as executable by the operating system; CGI scripts do not.

62. What is `htpasswd`?
 A. A password file for temporary passwords
 B. A program for cracking passwords to gain entry to a machine
 C. A way of connecting to several Apache servers using the same username and password combination
 D. An application provided by Apache that automates the generation of username/password files to be used with Web-based authentication

63. What is `mod_auth_mysql`?
 A. A module that allows Apache to pull dynamic content from a MySQL database for use on a Web page
 B. A module that can be used to put usernames and passwords into a MySQL database, rather than a text file
 C. A module that allows MySQL to store data in plain text
 D. A module that disables remote logins to the Web server

64. When a user fills out a username and password in an authentication window, how is that data passed back to Apache? How is it passed back to Apache+SSL?
 A. If passed to Apache alone, it is passed as encrypted data. If passed through Apache+SSL, it is clear text.
 B. If passed to Apache alone, it is passed as clear text. If passed through Apache+SSL, it is encrypted data.
 C. If passed to Apache alone, it is tunneled through ssh. If passed through Apache+SSL, it is tunneled through Telnet.
 D. All of the above.

65. Which of the following is true concerning content negotiation in Apache? (Select the best answer.)
 A. It is based on PHP user preferences.
 B. It is based on client preferences and resources available to the Apache server.
 C. It is used to set up a secure path between client and server.
 D. It is used for security only.

66. What are the pros and cons of style sheets?
 A. Pro: centralization of content. Con: browser incompatibilities.
 B. Pro: tighter control of text, graphics, and other page elements. Con: works only with Lynx.
 C. Pro: decentralization of content. Con: uses the XML standard.
 D. Pro: it is interchangeable with DSSSL. Con: uses the XML standard.

67. What is meant by "carrying state" in a Web browser?
 A. The browser is dependent on its locale.
 B. The browser sets up a TCP/IP stream between the client and server.
 C. The server keeps track of the client as it moves through the site.
 D. The client keeps track of the Web servers as it negotiates content.

68. Which of the following best describes the difference between Java and JavaScript?
 A. JavaScript is a programming language, and Java is only a protocol.
 B. Java is a compiled language, and JavaScript is an interpreted scripting language.
 C. JavaScript is embedded byte-code and requires a stand-alone JVM; Java does not.
 D. JavaScript is a server-side scripting language similar to PHP, and Java is embedded HTML.

69. How can you set up Apache to be used as a proxy?
 A. Load the squid module, `mod_squid`
 B. Load the proxy module, `mod_proxy`
 C. Use `winproxy`
 D. Redirect the port from 80 to 8080

70. Which of the following is true concerning proxy servers?
 A. They allow hosts to masquerade as other hosts.
 B. They cache dynamic content only.
 C. They cache frequently accessed Web pages.
 D. They allow hosts to connect to Web sites directly.

71. How does Apache use a caching proxy?
 A. It intercepts a request, compares it to source, and returns from cache, if available.
 B. It contacts InterNIC for previously cached material.
 C. It redirects the request to browsers on the same LAN that have already accessed a page.
 D. It downloads popular Web sites and replicates them locally.

72. Which of the following best describes a Server-Side Include?
 A. A scripting language that runs on the server, such as jsp, PHP, or asp
 B. A convenient way to handle data that is repeated multiple times by referencing one copy of the data
 C. A servlet engine
 D. None of the above

73. What is generally the file extension for files that use SSI?
 A. `.ssi`
 B. `.SSI`
 C. `.shtml`
 D. `.ssl`

74. Which of the following best describes URL rewriting?
 A. Dynamic HTML coding
 B. IP spoofing
 C. A rules-based rewriting engine that changes a requested URL to another URL
 D. An HTML debugger that can also debug CGI scripts

75. What is a phase?
 A. A chunk of CPU cycles for serving a document
 B. A main flow of operation in Apache
 C. A single `httpd` process
 D. A Web partition

76. What is a module?
 A. A bit of code written to handle a specific task, such as logging or authentication
 B. A Virtual Host specified by `named`
 C. The name for information kept between `<directive>` `</directive>` braces
 D. A `cgi` executable

77. What is a handler?
 A. An interrupt-driven Web site
 B. An interrupt-driven program
 C. A set of C functions inside a module
 D. A function that backs out of exceptions thrown by the Web server

78. Which of the following are phases of Apache processes that handle a request?
 A. URI to file name translation
 B. Authentication
 C. Authentication access checking
 D. Non-authentication access checking
 E. All of the above

79. What is the best way to prevent unused services from being compromised?
 A. Run a virus checker frequently
 B. Disable them
 C. Firewall effectively
 D. Do not run `inetd` as root

80. Why should you use PAM?
 A. It is an effective replacement for a firewall situation.
 B. It adds an extra layer of security.
 C. It allows all authentications to be handled by `mod_pam`.
 D. It prevents spoofing attacks.

81. What would you put in `/etc/hosts.deny` to prevent all computers from the bad.people.com domain from accessing the FTP service?
 A. `PREVENT: bad.people.com`
 B. `ALL: bad.people.com`
 C. `!bad.people.com`
 D. `EVERYTHING: *.people.com`

82. The index.html of Joe's home page has a mode of 600. Joe owns index.html. When others on the Internet try to access Joe's home page, they get `"Forbidden--permission denied"` errors. Which of the following will more than likely allow everyone to see Joe's single Web page?
 A. `apache --add index.html`
 B. `/etc/rc.d/init.d/httpd restart`
 C. `chown everyone index.html`
 D. `chmod a+rwx index.html`

83. The file `test` is a text file. What will the following command do?
```
md5sum test
```
 A. It will encrypt the file `test`.
 B. It will perform a checksum.
 C. It will produce an error.
 D. It will test the results of the `md5sum` command.

84. Which of the following are valid uses of custom logs? (Select three.)
 A. Looking for suspicious activity
 B. Looking for "page missing" errors
 C. Viewing site statistics
 D. Creating dynamic content
 E. Configuring Apache

85. What are some common tools that can be used to view logs? (Select three.)
 A. Analog
 B. Webalizer
 C. wusage
 D. logviewer
 E. SWAT

86. There is generally one log file that accommodates all virtual domains on a given system.
 A. True
 B. False

87. Dave has made some changes to his `httpd.conf` file. Which of the following is a valid way to make Apache reread its configuration file?
 A. `killall apache`
 B. `apachectl reload`
 C. `httpd restart`
 D. `kill -9 httpd`

88. Which of the following turns on Server-Side Includes for Apache?
 A. `Include SSI`
 B. `Options Includes`
 C. `SSI=yes`
 D. `Options SSI`

89. In `httpd.conf`, which of the following best describes what the `Include` directive does?
 A. It allows for other config files to be parsed.
 B. It controls all parsing.
 C. It logs all Apache activities, including parsing.
 D. It activates the parsing mechanism for Apache.

90. Which of the following can Apache *not* do when an error occurs?
 A. Display a message
 B. Allow the user to correct the problem
 C. Direct the user to another internal Web site
 D. Direct the user to another external Web site

91. In `httpd.conf`, which of the following best describes what `ServerRoot` does?
 A. It gives Apache the ability to send e-mail messages to system administrators.
 B. It specifies which documents Apache must read to redirect Web sites.
 C. It tells Apache where all the configuration and log files can be found.
 D. It defines the root name server for Apache.

92. The `.htaccess` file is parsed in which circumstance?
 A. Only when the Apache server is restarted
 B. Only when the Apache server is accessed
 C. Only after an Apache system is rebooted
 D. Only after an Apache module is installed

93. After Apache receives a request and looks at the appropriate data, it does which of the following?
 A. E-mails the system administrator
 B. Prompts the user for input
 C. Takes actions based on the configuration files
 D. Immediately starts up a Server-Side Include

94. By default, Apache listens on which of the following ports?
 A. 10
 B. 22
 C. 80
 D. 1024

95. By default, Apache checks which of the following locations for a user's Web pages?
 A. `/usr/local`
 B. `/username/public_html`
 C. `.htaccess`
 D. `apache.conf`

96. Which of the following tools is a benchmarking tool used to test the performance of Apache?
 A. bench
 B. strace
 C. apbench
 D. ab

97. The permissions must include `execute` if the file is to run as a CGI script.
 A. True
 B. False

98. Which of the following modules *must* be included with all Apache builds? (Choose two.)
 A. `mod_so.c`
 B. `mod_alias.c`
 C. `mod_cgi.c`
 D. `http_core.c`

99. To utilize the `apachectl fullstatus` option, which of the following must be present? (Choose two.)
 A. `cpuinfo`
 B. `hinv`
 C. `lynx`
 D. `mod_status.so`

100. Which of the following is true concerning cookies and Apache? (Choose two.)
 A. Users must accept cookies.
 B. Cookies can be set from CGI scripts.
 C. Cookies write small files to the client system.
 D. Cookies require the use of `mod_proxy.so`

Answers

1. **B.** `httpd.conf`

2. **A.** Scope is the area of effect for a given configuration.

3. **C.** A piece of code, separate from Apache, used to extend Apache's functionality

4. **A.** It is the process of resolving a name to its related IP address.

5. **D.** It is an order you give to Apache from within Apache's configuration files.

6. **B.** `/configure --enable-module=info`

7. **C.** `httpd.conf`

8. **A.** `AddModule modules/mod_log.config.o`

9. **A.** The Apache error log
 C. Try running Apache under strace.

10. **A.** Use http:// in the URL for the server.

11. **D.** It is a script used to start, stop, or restart Apache.

12. **A.** `srm.conf`
 C. `access.conf`

13. **B.** Net-based hosting

14. **C.** `--with-layout=Slackware`

15. **B.** False

16. **D.** 403

17. **A.** `make check`
 D. `make test`

18. **C.** `patch -pl < SSLpatch`

19. **A.** `<screen>Content-Type: text/html</screen>`

20. **B.** Perl, shell, C, C++

21. **D.** They are inoperable without `.htpasswd` files.

22. **B.** The `.htaccess` file

23. **D.** `Order Deny, Allow`

24. **D.** Typing, encoding, and language

25. **B.** No, run-time modules can be added using `apxs`.

26. **C.** Modules reduce the size of the Apache binary, so Apache consumes fewer resources.

27. **B.** Setting the proxy address and/or port number on the browser

28. **D.** XSSI can use conditional directives.

29. **A.** `<!--# SSI info -->`

30. **B.** It looks for the opposite of the conditional statement to be true.

31. **D.** Phase 6

32. **A.** Phase 3

33. **B.** Phase 4

34. **A.** ssh

35. **B.** `/etc/shadow`

36. **A.** `-%{User-agent}iÓ`

37. **B.** False

38. **A.** An IP address

39. **B.** It decreases performance because `httpd` has to search the whole tree if enabled.

40. **A.** It dumps a complete status page with the use of Lynx and `mod_status`.

41. **B.** Point your browser to http://127.0.0.1.
 C. Use the `rpm -qa | grep apache` command.
 D. Use the `find / -name httpd` command.

42. **A.** Its creator was Native American, of the Apache tribe.

 B. It was a series of patches to the original `httpd` daemon, hence "a patchy server."

43. **C.** A part of the request/response that contains information, which tells the client or the server what kind of information it is receiving

44. **A.** `rpm -Vp foo.rpm`

45. **A.** RPM
 B. DEB

46. **A.** `telnet localhost:80`, then type `GET /index.html`
 C. `lynx localhost:80`

47. **A.** It allows digital signatures.
 D. It requires encryption.

48. **B.** SSL allows Apache to handle the HTTPS protocol.
 C. Installing SSL with Apache requires the Apache+SSL patch.

49. **D.** A session key

50. **C.** `/configure --with-layout=gnu`

51. **A.** `/etc/rc.d/init.d/httpd start`
 C. `apachectl start`

52. **A.** `apachectl stop`
 D. `/etc/rc.d/init.d/httpd stop`

53. **C.** It specifies an IP address that should be used as a target for name-based virtual hosts.

54. **D.** It is a scripting language.

55. **D.** As a major mode of Emacs

56. **B.** PHP is essentially embedded HTML used for dynamic content.

57. **C.** Make the title of the page `–HelloÓ` and display `–Hello ClassÓ` on the page itself.

58. **A.** `<?` is a start tag, and `?>` is an end tag.

59. **C.** End the line with a semicolon `(;)`.

60. **A.** CGI is a method for allowing the Web server to execute a program, triggered by a Web page (or content). It allows for a more dynamic creation of content.

61. **C.** CGI scripts have to be recognized as executable by the operating system; PHP documents do not.

62. **D.** An application provided by Apache that automates the generation of username/password files to be used with Web-based authentication

63. **B.** A module that can be used to put usernames and passwords into a MySQL database, rather than a text file

64. **B.** If passed to Apache alone, it is passed as clear text. If passed through Apache+SSL, it is encrypted data.

65. **B.** It is based on client preferences and resources available to the Apache server.

66. **A.** Pro: centralization of content. Con: browser incompatibilities.

67. **C.** The server keeps track of the client as it moves through the site.

68. **B.** Java is a compiled language, and JavaScript is an interpreted scripting language.

69. **B.** Load the proxy module, `mod_proxy`.

70. **C.** It caches frequently accessed Web pages.

71. **A.** It intercepts a request, compares it to source, and returns from cache, if available.

72. **B.** A convenient way to handle data that is repeated multiple times, by referencing one copy of the data

73. **C.** `.shtml`

74. **C.** A rules-based rewriting engine that changes a requested URL to another URL.

75. **B.** A main flow of operation in Apache

76. **A.** A bit of code written to handle a specific task, such as logging or authentication

77. **C.** A set of C functions inside a module

78. E. All of the above

79. **B.** Disable them.

80. **B.** It adds an extra layer of security.

81. **B.** `ALL: bad.people.com`

82. **D.** `chmod a+rwx index.html`

83. **B.** It will perform a checksum.

84. **A.** Look for suspicious activity.
 B. Look for "page missing" errors.
 C. View site statistics.

85. **A.** Analog
 B. Webalizer
 C. wusage

86. **B.** False

87. **C.** `httpd restart`

88. **B.** `Options Includes`

89. **A.** It allows for other `config` files to be parsed.

90. **B.** Allow the user to correct the problem.

91. **C.** It tells Apache where all the configuration and log files can be found.

92. **B.** Only when the Apache server is accessed

93. **C.** Takes actions based on the configuration files

94. **C.** 80

95. **B.** `/username/public_html`

96. **D.** ab

97. **A.** True

98. **A.** `mod_so.c`
 D. `http_core.c`

99. **C.** `lynx`
 D. `mod_status.so`

100. **B.** Cookies can be set from CGI scripts.

Glossary

.cgi This extension describes CGI scripts.

access_log Apache adds a line to this log file for every request it receives.

access.conf A file for controlling access to documents.

Access control This involves using directives, such as `Allow`, `Order`, `Deny`, and `AllowOverride`, to control which users have access to the Web-space.

Access permissions Read, write, and execute are the possible permissions that a user or group can be given for a particular file.

AccessFileName This directive sets the name for the file that can control the settings for the `AllowOverride` directive.

Actions These work with handlers to extend the capabilities of Apache and PHP.

AddHandler This directive allows files with specified extensions to be mapped to a particular action.

AddLanguage Users designate what language a file is by adding a file extension to it and then using Apache's `AddLanguage` directive.

Alias An alternative name or label that refers to another particular name or label. The `Alias` directive creates the equivalent of symlinks for URLs.

Allow This directive, along with `Order`, controls who has access to the pages in the Web directory.

Allow changes The number of days before the user can change the password.

Anonymous access authentication Using the anon_auth_module, anonymous users can sign in with a default username and a valid e-mail address. These users can then access information.

Apache A popular, stable Web server, based on NCSA's `httpd`. It can be found at www.apache.org.

APACHE+SSL This is a version of Apache that has Secure Sockets Layer (SSL) security built into it.

apachectl This is the Apache HTTP server control interface; it helps control the `httpd` daemon.Fullstatus.

AuthGroupfile The file that associates group names with their members.

AuthName This access directive allows users to access password-protected areas, if they can complete the password challenge.

AuthType This access directive should be set to basic. Digest can be placed here to enable encrypted passwords.

AuthUserFile This directive points to the file that contains usernames and passwords.

BrowserMatch This directive defines environment variables based on the user-agent header.

Browsers A browser is an application that allows users to view and interact with information on the World Wide Web. PHP can tell a lot about a Web browser by reading the headers sent by the browser.

Buffer overruns The source of most attacks on Linux systems. These attacks occur when variables receive more data than the programmer anticipated or prepared for, leaving the system vulnerable.

cgi-bin The main directory for .cgi files.

checksum Information on a segment of data that can be checked against the data itself, in order to determine if the data arrived at its destination undamaged.

Child processes These are replications of Apache that dole out content and complete process requests.

Cipher Suite Negotiation This element of the handshake sequence allows the client and server to choose Cipher Suite that is supportable by both of them.

Common Gateway Interface (CGI) This allows Web users to interact with information servers, in order to provide a dynamic interface (as opposed to static).

Common Gateway Interface (CGI) scripts These are invoked through a Web server to provide for server-side, dynamic processing of data from users.

configtest Use the `configtest` command to have the `httpd` configuration files parsed without having to restart the server.

Container A container consists of a paired set of delimiters with the name of the scope to which the container applies.

Content negotiation The method in which Apache can make different languages or media types, which are found in a single resource, compatible, usually through an index page where a user can select an item.

Control structures These are statements in code that test conditions, such as the if, while, and for statements.

Cookies These are client-side files used to store data, sometimes allowing Web sites to record user actions without consent.

CookieTracking If `mod_usertrack` is compiled into Apache, `cookietracking on` will enable Apache to send a user-tracking HTTP cookie for all new requests.

Cryptography This is the science of information security. Cryptography deals primarily with the encryption and decryption of data.

CustomLog This directive is used by Apache to customize a log file format.

Daemon A program that runs in the background and interacts with the user only when called on.

Deny This directive can be used to prohibit access to specific clients.

Directives Directives can be grouped into containers that refer to a single directory, location, or virtual server in order to modify the default behavior set in the main body of `httpd.conf`.

DirectoryIndex This directive indicates the default files for a Web site in order of priority.

Domain Name System (DNS) This is an efficient, reliable, distributed system that allows the mapping of arbitrary names to IP addresses across a network.

Dynamic Shared Objects (DSO) These allow the user to add and remove features from Apache or to update a feature to a newer version, without recompiling the server.

Dynamic Web page A Web page that can change as it is updated from the server through a script or program; it can allow user interaction.

Encrypted signatures A method to send documents that are verifiable and undeniable.

Encryption This is any procedure used to convert plain text into scrambled text, in order to make it more difficult for unauthorized persons to intercept and interpret data.

Engine directive This directive dictates the directories or virtual servers that PHP should examine.

EOF End of File.

error_log Apache uses this to keep a log of all errors.

Extended SSI (XSSI) Expands on SSI by allowing conditions to be attached to directives, definition of variables by the user, and extension of external programs and CGIs.

Filtering The process of acting on the stream of content as it travels to or from the server, allowing a user to alter the data multiple times with different modules.

Forking When a stand-alone process splits into multiple processes.

Form handler Handles input into a form using GET or POST formats and generates appropriate response.

Forms Forms are used in scripting languages, such as HTML and PHP, to create data fields where the user can enter information.

Fragment attacks These attacks use the IP fragmentation feature to create very small packets that split the TCP header information into separate packets that, when reassembled, create a buffer overflow, which results in a denial of service.

Function This is a set of code that performs an action, which the user may want to call repeatedly. Use a function call to implement the function.

Group A group is a collection of users; groups help system administrators organize users with similar needs into manageable units. Each user is placed into at least one group, and additional group membership can be assigned in the `/etc/group` file.

Header A part of the request/response that contains information that tells the client or the server the type of information it is receiving.

htpasswd This is an application provided by Apache that somewhat automates the generation of username/password files to be used with Web-based authentication.

HTTP HyperText Transfer Protocol is the protocol that browsers and servers use to submit and server requests.

httpd A script that can start Apache's daemon (located in `/usr/sbin`).

httpd.conf Apache finds its settings in this configuration file. It has all the functionality of the former `access.conf` and `srm.conf` files.

HyperText Transfer Protocol (HTTP) The protocol that Web browsers and servers use.

include This directive can insert additional directives into the configuration files.

inetd Daemon started at boot-up that tells the system which sockets to listen for and what programs to start when those sockets are accessed.

IP address An identification code that uses four bytes of binary digits to distinguish a host on a network system.

KeepAlive When in effect, it causes child processes to spend time waiting for requests on an open connection.

Keep-Alive Allows persistent connections that help to send multiple requests over the same ICP connection.

Link A reference to a file or a directory.

LogFormat This directive is used by Apache to allow customization of the log file format.

logresolve This simple script comes with Apache and allows a user to turn IP addresses in the `access_log` file into resolved host names.

Lynx This is a widely used, text-mode Web browser. It can usually be found on any Linux distribution.

MD5 Reads data and calculates a cryptographic "checksum" that is very hard to duplicate, giving confidence that a file has not been intentionally modified.

mod_access Along with `mod_auth`, it provides the authentication functions of Apache.

mod_auth Provides the authentication functions of Apache and is used in combination with `mod_access`.

mod_perl A package that allows a user to write Apache modules completely in Perl.

mod_php A server-side, HTML-embedded scripting language that allows database requests through the Apache Web server.

mod_rewrite This module, found in Apache versions 1.2 and higher, uses a rule-based rewriting engine to rewrite requested URLs.

mod_ssl This module is derived from the Apache+SSL patch. It provides Secure Sockets Layer (SSL) security to Apache.

Module An Apache module is a piece of software, external to the main Apache program, which allows customization and additional options.

Multiviews Instructs Apache to look for a document that is the best match for a client's preferences.

MySQL An open source relational database management system that is frequently used for accessing, adding, and processing data that is in a database.

Name-based virtual hosting A virtual hosting scheme that allows more than one host to be run on the same IP address by adding names to the DNS as a `CNAME` on the machine.

NameVirtualHost This is how the user will specify on which IP address the server will take petitions for name-based virtual hosts.

Nobody This is a user with practically no privileges on a given system.

Opera This Web browser is smaller than popular browsers, such as Netscape, but is known for its stability and speed.

Order This directive, along with `Allow`, controls who has access to the pages in the Web directory.

Package A collection of files combined into a single file to simplify distribution and installation.

Packet A data block that is transmitted over a network or the Internet. A packet is made up of three pieces of data: the data being sent, the data that guides the packet along the way, and the data that fixes problems that arise during the process.

Packet filtering A router that allows the firewall to examine each packet of data against a set of predefined rules and determine whether to accept or deny the packet based on those rules.

Perl Program used on various platforms that integrates the best features of `awk`, `grep`, `sed`, and `tr`.

Phases Phases are one of the three critical components of the Apache API. The Apache flow of operation goes through eight phases each time Apache receives a request to serve a Web page, and a request flows from Phase 1 to Phase 8 in sequential order.

PHP A script language and interpreter that is primarily used with Linux Web servers. It provides flexibility for producing dynamic content.

Port A 16-bit number used to identify a network service on a host. Also, `Port` is a directive that can, when used in combination with BindAddress, specify an address and port, allowing different instances of the Apache server running on the same machine to listen for requests on different addresses or ports.

Port 80 The port most commonly used for a Web server or Hypertext Transfer Protocol daemon traffic.

Port-based virtual hosting These hosts can be used if another IP address or alias for the server is not available.

Pretty Good Privacy (PGP) Program used to secure communications, to verify that files have not been tampered with, and to verify that people are who they say they are.

Proxy This acts as a gateway for Web requests and services, permits and restricts access to a network based on a client's IP address, caches documents to help the system run more efficiently, and controls access (or helps firewalls control access) to networks.

Red Hat Package Manager (RPM) A program that provides flexibility in managing packages and is used in a number of Unix and Linux distributions.

Request This log file contains the actual HTTP request that was sent by a client, enclosed in quotes.

Rewrite Often used with `mod_rewrite`, this directive helps perform an internal file name-based subrequest. Rewrite can be used more than once.

RewriteCond A directive used with `mod_rewrite`, `RewriteCond` defines a condition for the `RewriteRule`. With one or more conditions preceding it, `RewriteRule` will not be used unless all these conditions apply.

Rewrite rule This directive can run multiple times. It defines a simple rule for rewriting.

Running herd A situation in which networks that use TCP/IP to communicate with their clients rapidly increase their offered loads.

ScriptAlias This directive is an important setting in the `srm.conf` file that gives an alias for the directory containing server-executed scripts.

Secure Sockets Layer (SSL) This protocol layer provides secure communication between client and server, by allowing mutual authentication, the use of digital signatures for integrity, and encryption for privacy.

ServerName This is the location where a host must be defined if it is going to have its own domain name associated with its IP address.

Server Side Includes (SSI) This directive can be used in HTML documents to echo information for any environmental variable.

ServerSignature This directive adds a line to logs with the server version number and the server name of the virtual host that generated the line.

ServerType Stand-alone Mode in the `ServerType` directive in which the `httpd` daemon listens for connections.

Setcookie This directive will use PHP to set a cookie in a Web browser, avoiding the JavaScript method.

Spoofing An attack in which packets are disguised to look as though they originally came from another source, often the internal network behind the firewall.

Squid caching proxy A program for caching HTTP information that does not require high-end hardware for significant performance increases.

srm.conf A file for specifying what kind of documents can be served.

Standalone This option is the default method for Apache. In stand-alone mode, servers are ready and waiting for connections. It is especially good for busy sites.

Static Web page A Web page that does not change.

Style sheets Describes to browsers how documents are presented on screens or in print.

Sudo Allows a system administrator to give certain users or groups of users the ability to run specific commands as root or another user while logging the commands and arguments.

Superuser The only account that may request certain system services such as changing date/time, adding a new user account, and increasing the priority of a process.

TCP/IP Transmission Control Protocol/Internet Protocol (TCP/IP) is the main protocol suite that allows all the computers on the Internet to communicate. Every computer is given a unique IP address.

TCP wrappers These have a set of wildcards that allow matching services and clients.

Thundering herd A problem in which all but one thread put themselves back on the wait queue to wait for the next connection.

Trojan horse A program that appears to do a certain task for the user, but its primary function is to perform another, possibly malicious, task.

Trusted networks Generally secure networks that employ one or more of the following: Apache+SSL, `mod_ssl`, password authentication, `.htaccess` file authentication, and anonymous access.

Uniform Resource Identifier (URI) A subset of the URL that identifies the path to the Web page.

Uniform Resource Location (URL) Part of the communication between clients and servers using HTTP.

Uniform Resource Name (URN) Part of the communication between clients and servers using HTTP.

User Anything that uses a process on a system.

UserDir A directive that assigns the name of the directory that is used as the individual's root document directory.

Virtual hosts Allow multiple domain names to be hosted under one IP address or multiple IP addresses with different domain names to be hosted.

Webuser/Webgroup `Webuser` and `webgroup` are the user and group that Apache will use to read files, execute CGIs, and create child processes.

Worms A form of attack that enters a computer system as a stand-alone program, not attached to other pieces of code, and may cause unintentional damage or resource depletion.

Index

A

ab (program), 99–100
Access, 146. *See also* Anonymous access
 blocking, 168–169
 checking. *See* Authentication; Non-authentication access checking
 control, 156–159, 167–170. *See also* Users
 testing, 158–159
 directives, 172–173
 method, 68
 permitting/restricting. *See* Client
AccessFileName, 168
Account information, 139
AddHandler directive, 39, 90
Add-on encryption, 136
address, 120–122
Alias, 42–43, 60, 89
Aliasing. *See* Internet Protocol
aliasnumber, 120
Allow (directive), 59, 156–158
AllowOverride, 61, 156, 172
American National Standards Institute (ANSI), 68
Analog, 110
Analysis tools. *See* Log
anon_auth_module, 159
Anonymous access, 137, 159
ANSI. *See* American National Standards Institute
Apache, 154–155
 API, introduction, 76–83
 basic configuration, 48–50
 child processes, 6
 compiling
 mod_ssl support, usage, 203–204
 verification, 204–205
 configuration, 161–162
 issues, 190
 conventions, 6–7
 core dump, discovery, 188–189
 current status, 5
 directories, layout, 6
 documentation, 51
 errors, log, 96
 files, layout, 6
 function, explanation, 5–10
 header files, 9
 history, 4
 initialization, 91–93
 installation, 29–34
 summary, 40–41
 testing, 33–34
 logging problems, 190–191
 logs
 configuration, 41–42
 viewing/interpretation, 109
 modules, 10
 introduction, 64–76
 obtaining, 10–11
 operations, theory, 3–18
 options directive, 53–54
 preparation, 58–63
 scripts, 35
 securing, 137–138
 server. *See* Perl-enabled Apache server
 downloading, 202–203
 setup, 45–54, 218
 source tree, 38
 SSL, implementation, 184
 starting, 26, 45, 146
 startup errors, 146
 stopping, 45
 TCP/IP, usage, 112–113
 user, 137–138
 version 1.3, 62
 HTTP server, 36
 version 2.0, 82–83
 vulnerabilities, 138–139
Apache configuration exercise, 207
 lab exercises, 210
 purpose, 207
 questions/answers, 210–211
 theory, 207–210
Apache custom log creation exercise, 239
 lab exercises, 241
 purpose, 239
 questions/answers, 242
 theory, 239–241
APache eXtenSion (APXS), 9
Apache proxy
 advantages, 144–145
 obtaining, 145–146
 server, 143–146
 setup, 159–164
 starting, 162–164
Apache Server, 145
Apache Tcl, 146
apachect1 fullstatus, usage, 88
apache-devel, 24
Apache-driven Web sites, 66
Apache+SSL contrast. *See* ModSSL

Apache+SSL installation exercise, 201
lab exercises, 202–205
purpose, 201
questions/answers, 205–206
theory, 201–202
Apache/Tomcat exercise, 223
lab exercises, 224–225
purpose, 223
questions/answers, 225
theory, 223–224
API. *See* Application Programming Interface
Append-only access, 165
Applets, embedding. *See* Java
Application Programming Interface (API), 76, 82
Application proxy servers, 149
APXS. *See* APache eXtenSion
ASP, 65
Attacks. *See* Counterattacks; Fragment attacks; Man in the middle attack; Source
handling, 152
response, 151–153
Authenticated user ID, 94
Authentication, 40, 80, 136–137. *See also* Clear text authentication; Digest authentication; htaccess; HyperText Transport Protocol; Passwords
access checking, 40, 81. *See also* Non-authentication access checking
Authentication--mod_access, 42–43
AuthGroupFile, 173
AuthName, 172–173
AuthType, 172
authuser, 41, 94
AuthUserFile, 173

B

Base systems, 18–34. *See also* Networking; Security; System administration
Basic redirect, 105–107
BCNF. *See* Boyce-Codd Normal Form
Benchmarking, 98–100
exercise. *See* Server benchmarking exercise
test, 107
tool. *See* HyperText Transport Protocol
Binary package, 29
BindAddress directive, 88
128-bit encryption, 37
Boot, usage. *See* httpd
Boot-time, starting, 146
Bounds checking, 142
Boyce-Codd Normal Form (BCNF), 67
BrowserMatch, 114
Browsers, 10
HTTP, interaction, 13–14
page, testing. *See* World Wide Web
type, 95
BSD systems, 163
Buffer overruns, 138, 141–142, 143
Bugtraq, 151
bytes, 42, 95

C

C, 223
functions, 78
programs, 143
C++, 15, 213
-c (concurrency option), 99
Caldera, 22
Case sensitivity, 128
CERT, 151
CGI. *See* Common Gateway Interface
cgi-scripting, 207
Checksums, 174–176
interaction. *See* Pretty Good Privacy
Child processes, 6. *See also* Apache
chroot (command), 57
Chunking, 115
Clear text authentication, 157
CLF. *See* Common Log Format
Client
access, permitting/restricting, 144
configuration, 229, 230
machine, 158
preferences, 12
Client/server architecture, 69
Combined Logfile Format, 110
Command-line options, 92
Commands, 177. *See also* Networking; Security; Shells; System administration
Common Gateway Interface (CGI), 60–61
bins, 155
configuration, 84
process, 90–91
explanation, 64
files, 80, 82
output, content, 100–102
page, 64
programming, 101
programs, 37
scripting, 35
security issues, 142–143
usage, timing, 64–66
Common Gateway Interface (CGI) scripts, 5, 8, 39, 78–79
creation, 100–107
creation exercise, 213
lab exercises, 214–215
purpose, 213
questions/answers, 215–217
theory, 213–214
example, 101–102
execution, 180
input data, 103
problems, 166
running, 165
writing, 100
Common Log Format (CLF), 41, 93
format, 95
support, 110
Compilation. *See* Source
Compile time, 43, 44
Compiled-in modules, 52
Compiler. *See* Platform-dependent compiler
flags, 29
Compiling, 30–33
mod_ssl support, usage. *See* Apache
Complexity, increase, 85
Comprehensive Perl Archive Network (CPAN), 76

Concurrency option. *See* -c option
Conditional statements, 181–183
CondPattern TestString, 127–131
configtest, 47
Configuration. *See* Server-side configuration
 files, 39–41, 43, 110, 139, 146
 editing, 163
 syntax, 146
 management, 56
 methods, 43–44
configure (method), 38, 39
Configure script, 32, 33. *See also* GNU configure script
Connection, 128–129
Connection-oriented network layer protocol, 183
Containers, 7
Content
 enabling, 170–172
 management, 56
 negotiation, 61–62
 overview, 11–18
Content-Length, 115
Control files, usage. *See* Defining
Core dump, discovery. *See* Apache
Corel, 196
 Linux, 22
Corporate intranets, 170
Counterattacks, 152–153
CPAN. *See* Comprehensive Perl Archive Network
Crypt_DES, 220
Cryptographic checksum, 175
Cryptographic security, 175
Cryptography, 183

D

Daemons. *See* httpd daemon; Multiple daemons
 configuration. *See* Multiple daemons; Separate daemons
 multiple instances, 209
Data
 backups, 71–74
 objects, 67
Database management system (DBMS), 66–67
Database-driven Web site, 58
Databases. *See* RPM; Structured Query Language
 concepts, 66–67
 design considerations, 69
 selection, 68–69
 setup. *See* MySQL
date, 41, 95
DBMS. *See* Database management system
DEB, 22
 packages, 45
Debian, 35, 195
Debian-based installations, 6
Declarative queries, 68
Decoy location, 84
Default index, 47–49
Defining
 control files, usage, 174
 httpd.conf, usage, 173–174
Demilitarized zone (DMZ), 150
Deny (directive), 59, 156–158
Destination address. *See* Internet Protocol
developer (option), 39
device, 120
Digest authentication, 136
Digests, 183
Directives, 7–8, 146. *See also* Access; Proxy
Directories, 64. *See also* Home directories
 enabling, 90–91
 indexes, 165
 layout. *See* Apache
 tree, 85
DirectoryIndex, 167
Distribution. *See* RPM-based distribution
 sites, 151
DMZ. *See* Demilitarized zone
DNS. *See* Domain Name System
Document
 caching, 144
 directory configuration, 53
 parameters, choice, 12–13
 root, usage, 91
Documentation, obtaining, 146
DocumentRoot, 53, 122
Domain name, mapping, 28
Domain Name System (DNS), 27, 144, 147, 150
 look-up, 60, 94. *See also* Double reverse DNS look-up
 name, 53, 199
 server, 112
 setup, 125
 usage, 28
Double reverse DNS look-up, 59
Downloads. *See* Source
Drives, usage. *See* Multiple drives
drop-table (option), 72
DSO. *See* Dynamic Shared Objects; Dynamically shared objects
Dynamic loading, 8
Dynamic Shared Objects (DSO), 6, 33, 199
 advantages/disadvantages, 9
 files, 9–10
 modules, 38
 support, 38, 228
Dynamically shared objects (DSO), 77

E

--enable-module=so, 33
Encoded passwords, 139, 176
Encoding, 11–12
Encryption. *See* Add-on encryption; 128-bit encryption
 algorithms, 184
 export laws. *See* U.S. encryption export laws
 scheme, 137
Environmental variable, 129, 177. *See also* REFERRER
EOF, 104
Error code, 82
Ethernet
 connection, 208
 interface, 120
Expirations, 114
eXtended SSI (XSSI), 179–183
 embedding, 181
ExtendedStatus, usage. *See* mod_status

F

-f (command-line option), 85
Files. *See* Configuration files; Dynamic Shared Objects; Log files
 authentication. *See* htaccess
 handles, 21
 including, 180–181
 layout. *See* Apache
 name, 12
 translation, 80
 security, 174
Filtering, 83. *See also* Packet filtering
Firewalls, 143, 146–153. *See also* Proxy
 machine, securing, 150–151
 network architecture, interaction, 130
 types, 147
Fixups, 40, 79, 81–82
FollowSymLinks, 60
Form handler, 103–104
Fragment attacks, 148–149
FTP, 58, 88, 147, 150, 201
 directory, 167
 server, 142, 148
 services, 135
fullstatus, 46–47
Functionality. *See* Modules
 local verification, 26–27
 verification, network usage, 27–28

G

Gateway
 interface. *See* Common Gateway Interface
 providing, 144
Gatewaying, 65
GET method, 103
Global Environment, 229
GNU configure script, 35
GNU GPL-license, 68
GNU software, 32
GNU standards, 36
graceful, 47, 91
Groups, 154–155, 176–177
GUI tool, 10

H

Handlers, 8, 78–79
Hard drives, 20
HARD_SERVER_LIMIT variable, 59
Hardware, 19–20
Hash function, 175
Headers, 112–113. *See also* HyperText Transport Protocol
 extension. *See* Netscape
 files. *See* Apache
 request. *See* Host
Hidden internal network, 1490
Higher-level protocols, 184
Home directories, 170–172
Host. *See* Internet Protocol; Multiple hosts; Non-name-based hosts
 header request, 115
 name, 126
 look-ups, 59–60
 operating system, 116
 request, directing. *See* Virtual host
host, 41, 94
Host Header, 122
Hostile programs, 138–142
Hosting. *See* Single daemon; Virtual hosting; World Wide Web
hosts.allow, 135–136
hosts.deny, 135–136
htaccess, 168
 file authentication, 137
HTML. *See* HyperText Markup Language
html. *See* UserDir
HTTP. *See* HyperText Transport Protocol
httpd, 4, 26
 binary, 52
 configuration, 228
 files, 47
 manual start, 45–47
 processes, number, 88
 runtime flags, 44
 starting, boot usage, 45
httpd daemon, 34–45, 84, 115
 customization, 35–36
 defaults, 34–35
httpd.conf, 7, 43
 configuration file, 161
 file, 41, 45, 87–91
 directives, 50
 usage, 49. *See also* Defining
httpd.conf Listen (directive), 84
%(HTTP:header), 130
HUP signal, 90
HyperText Markup Language (HTML), 101, 102
 code, 66
 document, 177, 181, 213
 file, 8, 81, 112, 182
 headers, 81
 input forms, 103
HyperText Transport Protocol (HTTP), 112, 201
 404 error, 241
 connection, 94, 229
 definition, 113–115
 Distributed Authoring and Versioning, 82
 headers, 128
 extension. *See* Netscape
 interaction. *See* Browsers
 MIME-header name, 130
 proxy
 cache, 144
 caching, 227
 field, 230
 request, 42, 95, 96, 227
 header, 123
 server, 58, 64, 213–214
 benchmarking tool, 243
 control interface, 244
 serving, 59
 status, 82
 code, 42, 95, 96
 usage, 13, 111
 user authentication, 217
 version 1.1, 113–115

I

IANA, 147
IBM DB2, 68
ICMP, 148
 message type, 148
Ideal permissions, matrix, 167
ident, 94
Identity check, 41
IETF, 184
ifconfig command, 120
Include (directive), 119
Includes. *See* Server-side includes
Indexed Sequential Access Method (ISAM), 66
Indexes (option), 47
index.html, 167
inetd file, 150

inetd (server process), 88
Informix SE, 68
Inline scripting languages, 65
Inodes, 21
INPUT tag, 103
Installation. *See* Apache; Knowledge matrix; Packages; RPM
 query. *See* Preinstallation query
Installation exercise, 195
 lab exercises, 196–199
 purpose, 195
 questions/answers, 199–200
 theory, 195–196
Internal network. *See* Hidden internal network
Internals. *See* Server
Internet Explorer, version 4.0b2, 114
Internet packets, 141
 header, 147
Internet Protocol (IP)
 aliasing, 119
 destination address, 148
 IP-based hosting, 64, 87, 118
 IP-based hosts, 63
 IP-based virtual hosting, 115, 119–122, 209
 masquerading, 147
 proxy servers, 149
 method, 121
 network, 28
 range, 162
 source address, 148
 tunnel, 148
Internet Protocol (IP) address, 27, 28, 63–64, 81. *See also* Virtual hosting
 access, 87
 blocking, 152
 conversion, 110
 defining, 84
 usage, 85, 120–124
 utilization, 149
Internet Service Provider (ISP), 57
Intranets. *See* Corporate intranets
I/O bandwidth, 13
Iogresolve, 110
IP. *See* Internet Protocol
IPsec, 136
ISAM. *See* Indexed Sequential Access Method
ISP. *See* Internet Service Provider
issue files, 136

J

Jakarta-Tomcat, 223, 224
Java, 15–17, 65
 applets, embedding, 15, 16
Java Development Kit (JDK), 223
Java Runtime Environment (JRE), 223
Java Virtual Machine, 223
Java-Apache, 145
Java-enabled Web browser, 17
JavaScript, 17–18, 65, 126, 208
 code, 17
JDK. *See* Java Development Kit
JPEG, 81
 images, 112
JRE. *See* Java Runtime Environment

K

KeepAlive, 63
Keep-Alive, 114
Kernel, 20–21, 119. *See also* Stock kernel
Knowledge matrix, installation/configuration, 3

L

-l (command-line option), 92–93
Labs/exercises, 193
%(LA-F:variable), 130
Language, 12–13
%(LA-U:variable), 130
Linker flags. *See* Platform-dependent linker flags
Linux, 9. *See also* Corel
 database systems, 68
 file system, 71
 IP masquerading host, 149
 optimization, 58
 preparation, 18–21
 programs, 81
 security model, 168
 system, 99, 164
Listen (directive), 117
LoadModule (command), 77
Local authorities, notification, 152
Location header, 105
Log
 analysis tools, 109–110
 analyzer, 110
 configuration. *See* Apache
 creation. *See* Apache custom log creation exercise
 keeping. *See* Transactions
 usage. *See* Multiple logs
Log files, 21, 75, 93–95, 146, 165
 formats, 95–98
 design, 96–98
 separation, 190
Logging. *See* Real-time logging; Request
 introduction, 83–84
 problems. *See* Apache
 process, 8, 20
Look-ups. *See* Host
Loopback
 device, 25
 interface, 120

M

Mailing lists. *See* PHP
Man in the middle attack, 141
Masquerading. *See* Internet Protocol
 host. *See* Linux
MaxClients, 59, 88
 150, 51
MaxRequestPerChild 0, 51–52
MaxSpareServers 10, 51
McCool, Rob, 4
MD5
 Digest Authentication, 83
 explanation, 175–176
Message type. *See* ICMP
MIME type, determination, 40
Mime types, 91
 determination, 81
MIME-header name. *See* HyperText Transport Protocol
MinSpareServers 5, 51
mod_access, 43

mod_auth, 42–43
 module, 176
mod_auth_db, 80, 83
mod_auth_digest, 83
mod_auth_mysql, configuration/running exercise, 217
 lab exercises, 218–221
 purpose, 217
 questions/answers, 221–222
 theory, 217–218
mod_autoindex, 40
mod_charset_lite, 82
mod_dav, 82
mod_file_cache, 82
mod_mime, 81
mod_perl, 76, 117, 146
mod_php, 117
mod_php4, 66
mod_proxy, 145, 146, 227
 installation, 228
 module, 161
mod_rewrite, 125–131
mod_ssl, 37, 183, 184
 support, usage. *See* Apache
ModSSL, Apache+SSL (contrast), 183–185
mod_status, 107–108
 exploitation, ExtendedStatus (usage), 108
 module, 46, 52
 usage, 62
Modules, 8–10, 36, 43, 77. *See also* Dynamic Shared Objects
 addition, RPM usage, 26
 advantages/disadvantages. *See* Static modules
 downloading, 196–197
 enabling, 37–39
 functionality, 37
 improvements, 82–83
 introduction. *See* Apache
 loading, 146
 performance, 37
Monitoring. *See* Performance monitoring
motd files, 136
Mozilla, 10
Multiple daemons, 84–85
 configuration, 85–88
 running, 86–87
 verification, 87–88
Multiple drives, usage, 20
Multiple hosts, 115–116
Multiple logs, usage, 20
Multi-user support, 68
MultiViews, 13
 option, 171
MySQL, 64, 66–76, 220
 architecture, 69
 command syntax, 71
 database, setup, 218
 server, 71
 daemon, 75
 student resources, 221
 system administration, 70
mysqldump (command), 71–73

N

-n (option), 99
Name
 look-ups. *See* Host
 translation. *See* Files
Name-based hosting, 118
Name-based hosts, 64
Name-based virtual hosting, 115–116, 122–125, 209
NameVirtualHost (directive), 63
NAT. *See* Network address translation
NCSA, 4
 version 1.3 Web server, 49
Netscape, 114, 183
 Refresh HTTP header extension, 106
 version 4.7x, 10
Network. *See* Hidden internal network; Internet Protocol
 access
 control, 144
 providing, 144
 addresses, 64
 architecture, interaction. *See* Firewalls
 layer protocol. *See* Connection-oriented network layer protocol
 security policy, 143
 usage. *See* Functionality
Network address translation (NAT), 147, 149
Networking, 111
 base systems, 116–131
 commands, 125–131
 operations theory, 111–116
 shells, 125–131
NIC, setup, 119
NIS domain, 150
Non-authentication access checking, 40, 79, 81
Non-CGI environment, 106
Non-dictionary words, 151
Non-name-based hosts, 63
Non-privileged programs, 138
Non-signed messages, 175
Normalization. *See* Tables

O

Online troubleshooting resources, 187–188
Open Source, 201
Openssl files, 203
Opera, 114
Operations theory. *See* Apache; Networking; Security; System administration
OPTIONAL trailer, 115
Oracle 8i, 68
Order (directive), 156–158, 169–170

P

Packages
 installation, 198
 introduction, 21–29
 validation, 22–23
Packet filtering, 147–148
 usage, reasons, 148–149
Packet, incoming/outgoing interface, 148
Parent process ID, 91
Password. *See* World Wide Web
 authentication, 137, 176–177
 method, 136
 combinations, 80
 entering, 173
 protection, 107, 153–154
 setting, 158
Passwords, 138. *See also* Encoded passwords
Pentium-based machine, 19
Performance monitoring, 107–109, 154
Perl, 64, 213, 223

code, 76
interpreter, 76
script, 65, 208
Perl-enabled Apache server, 76
Permissions, 154–156, 164–165
matrix. *See* Ideal permissions
PGP. *See* Pretty Good Privacy
Phases, 78–79
detail, 80–82
PHP, 64–66, 106, 146, 153, 208
mailing lists, 66
script, 82
Plain text, 220
Platform-dependent compiler, 10
Platform-dependent linker flags, 10
Port 80, 50
Port-based virtual hosting, 124
POST method, 103–104
PostgreSQL, 69
Preinstallation query, 197–198
Pretty Good Privacy (PGP), 136
checksums, interaction, 175
Private Documents, 172–173
Privileges, 74–75
PRNG. *See* Pseudo-random number generator
Problem Report (PR), 187
Process
creation, 62
death, 62
ID, 95. *See also* Parent process ID
Proxy. *See* Apache proxy
server. *See* Apache proxy; Application proxy servers; Internet Protocol; SOCKS
firewalls, 149
types, 149
usage, 161–162
serving, 147
specific directives, 159–161
Proxy configuration exercise, 227
lab exercises, 229–230
purpose, 227
questions/answers, 230–231
theory, 227–229
Proxy/Web server, usage, 162
Pseudo-random number generator (PRNG), 175
Pseudo-random numbers, 175
public_html. *See* UserDir
Python, 213

Q

Query optimization, 68

R

RAM. *See* Read Access Memory
Random character generator, 138
RC4, 136
RDBMS, 68
Read Access Memory (RAM), 19, 20
adding, 59
README file, 32, 48
Read-only directories, 87
Real-time logging, 191
Red Hat, 195
CD-ROM RPM, 29
layout, 53
servers, 22
version 6.2, 24, 45, 86
Redirect. *See* Basic redirect
REFERRER, environmental variable, 239
Request, 128–129. *See* Host; HyperText Transport Protocol
directing. *See* Virtual host
logging, 40, 82
phases, logging, 79
processing, 39–40
request, 42, 95
Request For Comment (RFC) 1413, 41, 94
Require (directive), 173
Resources, usage, 85
Response
header, 95
sending, 40, 82
restart method, 91
RewriteRule, 60, 130
Rewriting. *See* URL engine. *See* Rule-based rewriting engine
RFC. *See* Request For Comment
Root access, 33
Root user, 53
Root-level access, 141
Rootshell, 151
RPM. *See* Source
database, 23
installation, 24
package, 45
file, 23
security tasks, 22–24
RPM-based distribution, 24, 26
RPM-based installations, 6
RSA, 136
RSA Data Security, 185
RSAREF code, 185
Rule-based rewriting engine, 125
Run-level script, 163
Runtime flags. *See* httpd

S

safe_mysqld (script), 70
Satisfy (directive), 157, 169
ScriptAlias (directive), 39, 89
Scripting, 165–166. *See also* Common Gateway Interface
mistake, 101
Scripts. *See* Apache; Common Gateway Interface; Configure script; GNU configure script
execution, 181
SCSI drives, 20
Search string, 25
Secure Sockets Layer (SSL), 36–37, 136, 183–184
certificates, 134
contrast. *See* ModSSL
implementation. *See* Apache
legal issues, 184–185
Security, 133. *See also* Cryptographic security; Files
base systems, 154–174
commands, 174–177
concerns, 134
fundamentals, 164–167
holes, 165

Security, *Continued*
issues. *See* Common Gateway Interface
operations theory, 133–154
policies, 134–136. *See also* Network
privileges, management, 74–75
profile, 69
risks, 63
shells, 174–177
system utilities, 177–185
tasks. *See* RPM
tightening, 131
updates, 22
locating, 25
verification, 23
Separate daemons, configuration, 117–118
Server. *See* Apache; Application proxy servers; Internet Protocol; SOCKS proxy server
downloading. *See* Apache
environment, 51, 102–103
internals, 129
operating system, 89
referencing, 105
setup, 198–199
starting/stopping, 70–71
variables, 128–130
Server benchmarking exercise, 243
lab exercises, 245–246
purpose, 243
questions/answers, 247
theory, 243–244
ServerAdmin root, 52–53
ServerAlias, 122
server-info, 109
Server-info page, 109
ServerName, 53, 122
ServerPath, 122
Server-side configuration, 161–164
Server-side includes, 81, 177–179
Server-Side Redirects, 81
ServerSignature, activation, 84
ServerType stand-alone, 50–51
Service provider, notification, 152
Shadow copy, 141
Shadow Suite, 138–139
Shareware, 110
Shells. *See* Networking; Security; System administration
commands, 179–180
script, 65
scripting, 213
Signatures, 183
Single daemon
running, 34
virtual hosting, 118–119
Slackware, 163, 195
SMTP traffic, 148
SOCKS, 143
proxy server, 149
Solaris, 62
version 2.3, 191
Source
attacks, 148–149
code installations, 30
compilations, 34
downloads, 34
form, 40
RPMs, 29
tree. *See* Apache
Source address. *See* Internet Protocol
Spec file, 29
Special variables, 129
Spoofing, 138, 141, 148. *See also* Transport Control Protocol
SQL. *See* Structured Query Language
Squid log format, 110
SSI, 65, 177–179
SSL. *See* Secure Sockets Layer
Stand-alone. *See* ServerType stand-alone
process, 89
StartServers, 88
Static modules, 77
advantages/disadvantages, 9
status, 42, 95
Stock kernel, 20
Stock vendor builds, issues, 22
Structured Query Language (SQL). *See* MySQL
code, 66
commands, 72
databases, 68
statements, 72
Student resources. *See* MySQL
Stuffit archives, 12
Style sheets, 14–15
Substitution, 130–131
sudo (command), 57
suExec, 166
suexec (command), 57
Summary, 110
SunOS, 62
Swap space, usage, 20
SymLinksIfOwnerMatch, 60
System
administrator, 56
calls, 143
memory, 19
resources, 100
restoration, 152
utilities, 34–54. *See also* Security; System administration
variables, 129
System administration, 55. *See* MySQL
base systems, 84–98
commands, 98–100
operations theory, 56–84
shells, 98–100
system utilities, 100–110
SystemV systems, 163, 182

T

-t (command-line option), 92
Tables, normalization, 67
Tarballs, 22
unpacking, 29, 30
TCL, 213
TCP. *See* Transport Control Protocol
TCP/IP. *See* Transport Control Protocol/Internet Protocol
Telnet, 88
services, 135
usage, 28
Test results, 23
Text editor, 163
Text files, 112
Third-party tools, 68
TLS. *See* Transport Layer Security
Tomcat. *See* Jakarta-Tomcat

exercise. *See* Apache/Tomcat exercise
Top-level directory, 30
Transactions, logs (keeping), 75–76
Transport Control Protocol (TCP), 148
 connection, 94, 114
 spoofing, 141
 wrappers, 135, 152
Transport Control Protocol/Internet Protocol (TCP/IP), 183
 explanation, 112–113
 stack, 21
 version, 59
Transport Layer Security (TLS), 184
tripwire (tool), 152
Trojan Horses, 139, 140
Troubleshooting, 187
 resources. *See* Online troubleshooting resources
 sites, 189–190
Tuning, 58
Type, 11

U

UDP. *See* User Datagram Protocol
Uniform Resource Identifier (URI), 112, 125–126
Uniform Resource Locator (URL), 171
 pattern, 130
 rewriting, 36, 125–131
 Rewriting Guide, 126
 specification, 181
Uniform Resource Locator (URL) rewriting exercise, 233
 lab exercises, 236–237
 purpose, 233
 questions/answers, 237–238
 theory, 233–236
Uniform Resource Name (URN), 112
University installations, 170
Unix
 security model, 168
 system administrator, 56
Updates. *See* Security
Upper-level protocols, 201
URI. *See* Uniform Resource Identifier
URL. *See* Uniform Resource Locator
URN. *See* Uniform Resource Name
U.S. encryption export laws, 136
USENET, 188
User Datagram Protocol (UDP), 148
UserDir
 disabled root, 53
 html, 171–172
 public_html, 53
Usernames, 80, 157
Users, 154–155, 176. *See* World Wide Web
 access control, 167–170
 accounts, management, 74–75
 authentication. *See* HyperText Transport Protocol
 ID, 117, 155, 164. *See also* Authenticated user ID
 requirements, 139
/usr/local directory, 32
Utilities, 139. *See also* Security; System; System administration

V

-v (verbose option), 99
/var/apache/logs/access_log, 93–94
/var/apache/logs/error_log, 96–98
Variables, 177–179. *See also* Server; Special variables; System
VBScript, 65
Verbose option. *See* -v option
Virtual host, 121
 container, 85
 request, directing, 118
Virtual Host documentation, 123
Virtual hosting, 116–118. *See also* Internet Protocol; Name-based virtual hosting; Port-based virtual hosting; Single daemon
 examples, mixed methods (usage), 124–125
 introduction, 63–64
 one IP address, example, 122–123
Virtual system, 57
VirtualHost
 192.168.0.3, 121
 tag pair, 63
/VirtualHost, 122
Viruses, 139, 140

W

Walk-throughs, 159
W3C. *See* World Wide Web Consortium
Webalizer, 110
Webmaster, 4, 56–58
--with-layout=GNU, 36
--with-layout=RedHat, 33
World Wide Web Consortium (W3C), 64, 106
World Wide Web (WWW/Web), 37, 141, 147
 browser, 10, 13, 64, 112. *See also* Java-enabled Web browser
 page testing, 205
 clients, 83
 concepts, 5
 hosting, 3
 managers, 57
 pages, 90
 password, 172
 server, 11, 105
 daemon, 56
 usage. *See* Proxy/Web server
 sites, 6, 13. *See also* Apache-driven Web sites
 hosting service, 58
 users, 5
 Web-accessible method, 107
 Web-based administration, 110
 Web-based user, 220
 Web-space, 156, 157
Worms, 139, 141
wu-ftp xferlog, 110
Wusage, 110
www/hosts/ipbased1/htdocs, 122
www.iptest.com, 122

X
X Window system, 10
XML, 145
XML-Apache, 145
XSSI. *See* eXtended SSI

Z
Zip files, 11